UNIVERZA V LJUBLJANI
FAKULTETA ZA RAČUNALNIŠTVO IN INFORMATIKO

Saša Nebojša Potežica

RAZVOJ APLIKACIJ ZA PAMETNE TELEFONE

DIPLOMSKO DELO UNIVERZITETNEGA ŠTUDIJA

Mentorica:
doc. dr. Mojca Ciglarič

Ljubljana, 2009

Namesto te strani vstavite original izdane teme diplomskega dela s podpisom mentorja in dekana ter žigom fakultete, ki ga diplomant dvigne v študentskem referatu, preden odda izdelek v vezavo!

I Z J A V A O A V T O R S T V U

diplomskega dela

Spodaj podpisani/-a **Saša Nebojša Potežica**

z vpisno številko **63010116**

sem avtor/-ica diplomskega dela z naslovom:

Razvoj aplikacij za pametne telefone

Smartphone Application Development

S svojim podpisom zagotavljam, da:

- sem diplomsko delo izdelal/-a samostojno pod mentorstvom (naziv, ime in priimek)

  doc. dr. Mojce Ciglarič

  in somentorstvom (naziv, ime in priimek)

  _____

- so elektronska oblika diplomskega dela, naslov (slov., angl.), povzetek (slov., angl.) ter ključne besede (slov., angl.) identični s tiskano obliko diplomskega dela

- soglašam z javno objavo elektronske oblike diplomskega dela v zbirki »Dela FRI«.

V Ljubljani, dne **15.09.2009** Podpis avtorja/-ice: _____

Zahvala

Zahvaljujem se mentorici doc. dr. Mojci Ciglarič za mentorstvo in nasvete pri izdelavi diplomskega dela.

Zahvaljujem se svoji družini za podporo in potrpežljivost v času študija.

Svojim prijateljem in prijateljicami se zahvaljujem, ker so me spodbujali v času pisanja diplome.

Kazalo

Seznam kratic

IaaS	*(angl.) Infrastructure as a Service;* Infrastruktura kot storitev.
PaaS	*(angl.) Platform as a Service;* Platforma kot storitev.
SaaS	*(angl.) Software as a Service;* Programska oprema kot storitev.
GPS	*(angl.) Global Positioning System;* Sistem za pozicioniranje.
PDA	*(angl.) Personal Digital Assistant;* Osebni organizator.
NDK	*(angl.) Native Development Kit;* Orodje za domoroden razvoj aplikacij.
CLDC	*(angl.) Connected Limited Device Configuration*; Konfiguracija za omejene naprave.
MIDP	*(angl) Mobile Information Device Profile;* Profil za informacije o mobilni napravi.
CIL	*(angl.) Common Intermediate Language*; Skupni vmesni jezik.
CLR	*(angl.) Common Language Runtime*; Skupni izvajalni jezik.
CSS	*(angl.) Cascading Style Sheets*; Prekrivne predloge.
HTML	*(angl.) HyperText Markup Language;* Označevalni jezik za oblikovanje večpredstavnostnih dokumentov.
RAD	*(angl.) Rapid Application Development;* Hiter razvoj aplikacijs.
XML	*(angl.) Extensible Markup Language;* Razširljivi označevalni jezik.
JDK	*(angl.) Java Development Kit;* Orodja za razvoj v Javi.
SKD	*(angl.) Software Development Kit;* Orodja za razvoj aplikacij.
MID	*(angl.) Mobile Internet Device;* Prenosna naprava za dostop do interneta.
JSON	*(angl.) JavaScript Object Notation;* Notacija za opis objektov v JavaScript.
SAX	*(angl.) Simple API for XML;* Programski vmesnik za XML.

Povzetek

Pametni telefoni so majhne in zmogljive naprave. Ker v njih vidim prihodnost, sem se odločil raziskati razvoj na teh računalnikih v malem. Od vseh platform, ki so v tem trenutku aktualne, sem izbral tri po mojem mnenju najbolj perspektivne in za vsako od njih razvil primer aplikacije za branje spletnih dnevnikov.

V uvodnem poglavju predstavim naraščajoči trend pametnih telefonov in razvoja aplikacij zanje.

V drugem, teoretično najobsežnejšem poglavju razložim kaj so pametni telefoni ter predstavim zgodovino in načine razvoja za mobilne naprave. Sledi pregled operacijskih sistemov za pametne telefone in razvoja aplikacij za vsakega od njih.

V tretjem poglavju predstavim aplikacijo za branje spletnih dnevnikov, neodvisno od operacijskega sistema in pristopa. Navedem tudi možne izboljšave.

V četrtem poglavju aplikacijo razvijem za Android, iPhone in WebOS operacijske sisteme, hkrati pa opišem in na primerih kode predstavim razvojne koncepte in posebnosti. Na koncu razvoje primerjam ter povzamem prednosti in slabosti.

V zadnjem poglavju podam ugotovitve in sklep, do katerega sem prišel v okviru izdelave pričujočega diplomskega dela. Na kateri operacijski sistem sem se osredotočil in zakaj ter podam smernice nadaljnjega raziskovanja.

Ključne besede:
mobilna tehnologija
mobilni operacijski sistemi
razvoj
Android
iPhone

Abstract

Smartphones now days are compact and capable devices. Because of their vast potential in the future, decision was made to research application development for these miniature computers. Out of six platforms powering mobile devices today, three were chosen as most prosperous. For purpose of determining which mobile operating system holds most promise in the future, similar applications were developed for all three systems.

In introduction, increasing trend of smartphones and application development for their operating system is presented.

Second chapter clarifies term smartphone, describes history and types of development for mobile devices. It is followed by brief overview of mobile operating systems and procedures taken when developing applications for each system.

In third chapter application for managing blogs is explained independently of operating system and development approach. In the end, possible application improvements are listed.

Chapter four is focused on application development for Android, iPhone and WebOS platforms. Platform specific development concepts are described and displayed on basic code examples. It is followed by comparison and summary of platform's advantages and disadvantages.

Final chapter includes my findings and conclusion based on observations and practical work during this dissertation. It also defines the platform I focused on for future development and guidelines for future research.

Keywords:
mobile technology
mobile operating systems
development
Android
iPhone

1. Uvod

Mobilni telefoni postajajo pravi računalniki v malem. Njihova procesorska moč in komunikacijska povezava omogočata, da z njimi počnemo veliko več kot samo telefoniramo, medtem ko relativna majhnost in prenosljivost nudita prednost pred namiznimi in prenosnimi računalniki. Prav zaradi tega imajo ogromen potencial že danes, zagotovo pa lahko trdimo, da bo v bližnji prihodnosti mobilnih naprav še več. Posebna vrsta telefonov so pametni telefoni, ki se od običajnih ločijo po tem, da jih poganja zmogljiv operacijski sistem. Ta je za razliko od namenskih sistemov v telefonih naprednejši, nadgradljiv in omogoča namestitev aplikacij, ki funkcionalnost telefonov še dodatno razširijo.

Bistvena lastnost operacijskih sistemov, ki poganjajo pametne telefone je v tem, da lahko vsakdo, z nekaj predznanja o konceptih programiranja in programskih jezikih, napiše lastno aplikacijo za pameten telefon. Aplikacijo lahko uporablja zase, ali pa - kar je bolj smiselno - ponudi ostalim uporabnikom pametnih telefonov. To lahko stori preko namenskih distributerskih aplikacij in portalov, bodisi zastonj bodisi proti plačilu.

Danes smo v obdobju, ko proizvajalci telefonov ponudijo napravo, zmogljivo platformo, orodja za razvoj in kanal za prodajo napisanih aplikacij, dodano vrednost pa določajo razvijalci uporabniških aplikacij. Razvoj aplikacij za pametne telefone vse bolj pridobiva na pomenu, zato sem ga izbral kot osrednjo temo diplomskega dela.

Namen:

- seznaniti se s trgom pametnih telefonov in njihovimi operacijskimi sistemi
- raziskati razvoj in potrebno znanje za pisanje aplikacij
- določiti smiselnost razvoja aplikacij za mobilne naprave
- seznaniti se s pojmi, koncepti in specifikami razvoja za mobilne naprave
- olajšati izbiro pri nadaljnjem razvoju in specializacijo na določeno platformo

Cilji:

- na primeru aplikacije ugotoviti, katero razvojno okolje je primernejše za nove razvijalce
- podati sliko, o smiselnosti in prihodnosti razvoja za mobilne naprave
- ugotovitve prenesti na svoje bodoče delovno mesto

1.1. Vloga mobilnih naprav v računalništvu

V okviru pisanja diplomskega dela sem se srečal z nekaterim pojmi, ki so povezani tudi z mobilnimi napravami:

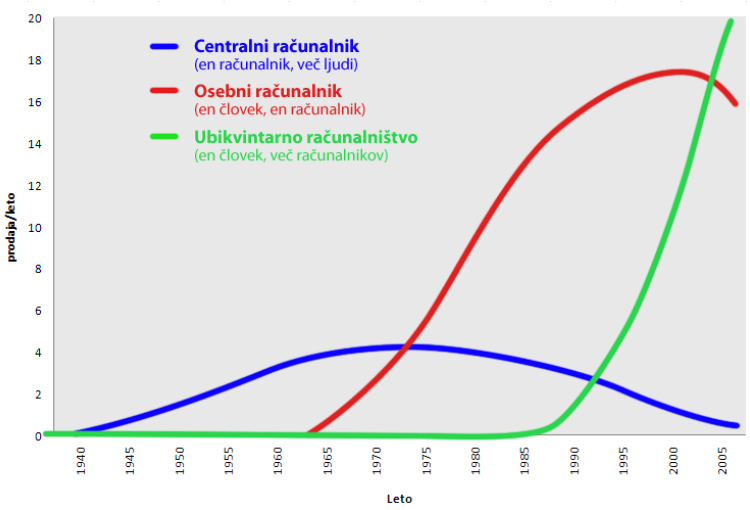
Obogatena resničnost *(angl. Augumented reality)*

Pri obogateni resničnosti gre za mešanje [1] realnega sveta s podatki oziroma računalniškimi učinki, ki so prikazani v človeku razumljivejši obliki. Večinoma gre za zajemanje slike s kamero v realnem času nato pa obdelavo posnetka. Na podlagi senzorjev ter prepoznavanja objektov dodamo računalniško ustvarjene dodatke. Ti so lahko preprosti v obliki informativnih napisov ali v naprednejši obliki s prikazom tri-dimenzionalnih objektov (recimo prikaz navidezne hiše v realnem okolju). Mobilne naprave so dobra platforma za razcvet obogatene resničnosti, saj so dostopne, zmogljive, nudijo širok nabor uporabnikov in konec koncev tudi možnosti za razvoj aplikacij, ki ponujajo obogateno resničnost. Kot primer lahko omenim Google Skymap, ki preko vgrajenega kompasa, merilca pospeškov in lokacijskih storitev na zaslonu v realnem času prikazuje proti kateremu ozvezdju, planetu, zvezdah in nebesnih znamenjih imamo obrnjen telefon in nas po njih tudi vodi.

Vsepovsod prisotno (ubikvintarno) računalništvo *(angl. Ubiquitous computing)*

Ubikvintarno računalništvo (skrajšano UbiComp) je najnovejši trend v računalništvu in tehnologiji. Gre za vseprisotnost naprav in zlivanje računalniške tehnologije z našim vsakodnevnim življenjem – tehnologija postaja nevidna, del nas – everyware [2].

Ubikvintarno računalništvo lahko vzamemo tudi kot nasprotje navidezne resničnosti [3]. Medtem ko navidezna resničnost skuša človeka postaviti v ustvarjen računalniški svet, ubikvintarno računalništvo postavlja računalniško tehnologijo v človeški svet.



Slika 1: Trendi skozi zgodovino računalništva

Slika 1 prikazuje tri trende skozi zgodovino računalništva. Osrednje računalnike so postopoma nadomestili osebni računalniki, v prihodnosti pa bo prevladoval trend vsepovsod prisotnih sistemov, kjer bodo računalniki neopazno integrirani z okoljem in posameznika spremljali povsod. Mobiteli so dober kazalec tega trenda, saj so naši stalni spremljevalci in vse manjši [4].

Tehnološka konvergenca *(angl. Technological convergence)*

Splošno jo lahko opredelimo kot proces v katerem se telekomunikacijska in informacijska tehnologija ter mediji zlivajo v eno. S tehničnega stališča pomeni možnost, da lahko infrastruktura prenaša kakršen koli tip podatkov, s funkcionalnega pa da lahko izdelujemo edinstvene naprave, ki so zmožne predvajati multimedijsko vsebino, prenašati podatke in izvajati procesorsko zahtevne operacije. Mobilne naprave so dober primer tehnološke konvergence.

Oblačno računalništvo *(angl. Cloud computing)*

Gre za novi model računalništva [5], ki se je razvil zaradi velike rasti števila naprav, ki so ves čas povezane na internet. Ogromen del procesiranja in shranjevanja podatkov se je lahko prenesel na zunanje računalniške mreže in gruče, do njih pa lahko dostopamo iz različnih lokacij v obliki terminalov. Obstajajo tri večje oblike storitev oblačnega računalništva:

- infrastruktura kot storitev (IaaS)
- platforma kot storitev (PaaS)
- programska oprema kot storitev (SaaS)

Od teh je za končnega uporabnika najbolj zanimiva programska oprema kot storitev. Ker so mobilne naprave strojno omejene bodo v prihodnosti še bolj izkoriščale oblačno računalništvo za shranjevanje podatkov in procesiranje zahtev.

Slika 2: Oblačno računalništvo (vir: Tomislav Rozman [6])

2. Pametni telefoni

2.1. Definicija

Mobilni telefoni danes niso več samo naprave, katerih glavna funkcija je telefoniranje. Postopoma so postajali zmogljivejši in začela se je brisati meja med telefonom, dlančnikom, predvajalnikom večpredstavnih vsebin, fotoaparatom in danes že računalnikom. Začenjajo nadomeščati naprave in pripomočke, ki nas spremljajo v vsakdanjem življenju in so postali obvezen spremljevalec naše kulture. Življenja brez telefona si večina ne more predstavljati, nekateri pa si sploh ne morejo več privoščiti.

Definicija pametnih telefonov [7] se od posameznika do posameznika razlikuje in jo je težko opredeliti. Za nekatere uporabnike so to naprave z značilnostmi, kot so dostop do interneta, odjemalec elektronske pošte; drugi pa si pod tem nazivom predstavljajo miniaturne računalnike, vključno z vmesniki ter vhodi in izhodi, ki imajo vdelano možnost telefoniranja.

Poleg tega se definicije pametnih telefonov, ki so vezane na funkcionalne in zmogljivostne značilnosti mobilne naprave, spreminjajo skozi čas, saj prihajajo nove tehnologije, ki kmalu postanejo del vsakdana. Zaradi tega definicijo pametnih telefonov ni smiselno vezati na programske, strojne ali funkcionalne lastnosti naprave.

Kar jih loči od običajnih telefonov je fleksibilen operacijski sistem, ki razvijalcem nudi platformo za enostaven in učinkovit razvoj novih aplikacij, ki bodo izkoriščale vse funkcionalnosti, ki jih naprava nudi. Uporabnikom pametni telefoni omogočajo namestitev novih aplikacij, ki napravi dodajo dodano vrednost, boljšo izkušnjo, povečano produktivnost in boljšo informiranost.

Današnji pametni telefoni nudijo:

- zmogljiv operacijski sistem, za katerega lahko razvijamo aplikacije
- brezžično povezavo
- popolno podporo elektronski pošti (nekateri podpirajo potisno pošto)
- sinhronizacijo s pošto, kontakti in koledarjem (Exchange, Lotus Notes)
- zunanjo ali virtualno QWERTY tipkovnico
- podporo dokumentom - PDF, Microsoft Office
- na dotik občutljiv zaslon (nekateri zaznajo več dotikov)
- senzorje kot so GPS, kompas, merilca pospeška in orientacije

2.2. Zgodovina

Veliko zaslug za današnje pametne telefone je potrebno pripisati dlančnikom. Za razliko od današnjih naprav, je bila v preteklosti ostra meja med dlančniki in telefoni. Slednji so bili namenjeni telefoniranju in kratkim sporočilom, medtem ko so bili tako imenovani osebni organizatorji namenjeni hranjenju in dostopanju do podatkov. Prvi osebni organizator je bil Casio PF-3000 leta 1983, medtem ko izraz *Personal digital assistant* (PDA), oziroma dlančnik zasledimo leta 1992, ko je Apple predstavil platformo Newton.

Razvoj dlančnikov ter telefonov je potekal ločeno in velja, da so z združevanjem telefonov s funkcionalnostmi dlančnikov prvi zametki pametnih telefonov. To je leta 1996 storilo podjetje Nokia s serijo Communicator 9000. Bili so telefoni z odprtim operacijskim sistemom, ki je omogočal nalaganje dodatnih aplikacij, kasnejši modeli pa so vsebovali brezžično povezavo do interneta, kamero in GPS modul. Lahko rečemo, da je po definiciji prvi pametni telefon.

Kljub temu pa kot prvi pametni telefon štejemo napravo Simon, ki jo je razvilo podjetje IBM in predstavilo leta 1992. Bil je prvi, ki je telefonu dodal funkcionalnosti osebnega organizatorja, pozivnika in faks naprave. Vseboval je za tiste čase za telefon še nevidene aplikacije, kot so koledar, svetovno uro, kalkulator, odjemalec za pošto in beležko. Interakcija je potekala preko na dotik občutljivega zaslona.

Po letu 2000 so začeli telefoni nadomeščati dlančnike, dlančniki pridobivati sposobnosti telefonov, oboji pa so bili v domeni poslovnih uporabnikov. Podjetjem kot sta Nokia in Ericsson se je v igri pametnih telefonov pridružilo podjetje RIM (Research In Motion) s serijo telefonov BlackBerry. Njihova posebnost je bil potisni mail *(angl. Push mail)*, ki je temeljil na njihovi lastni storitvi.

Leta 2002 je na področje mobilnih naprav vstopil Microsoft z lastnim operacijskim sistemom Windows CE, za dlančnike iz katerega so se kasneje razvili Windows Mobile za pametne telefone. To je bil prvi operacijski sistem za mobilne naprave, ki je bil namenjen telefonom različnih proizvajalcev.

Trg poslovnih uporabnikov je bil leta 2005 zasičen z dlančniki in pametnimi telefoni. Nokia je takrat naredila veliko potezo in ponudila pametne telefone za splošne uporabnike. To je bila N serija, ki jo je poganjal operacijski sistem Symbian in v prenovljeni verziji poganja tudi današnje Nokiine pametne telefone.

Trg pametnih telefonov je junija 2007 prevetril Apple z iPhone-om. Ta je bil med uporabniki zelo dobro sprejet in lahko trdimo, da je začel trend pametnih telefonov z na dotik občutljivim zaslonom. Večina današnjih proizvajalcev se zgleduje po iPhone-u in se skuša vsaj približati njegovemu uspehu.

Odgovor na iPhone je novembra 2008 podal Google, vendar ne z napravo, temveč s prvim povsem odprto kodnim operacijskim sistemom, imenovanim Android. Tako kot Windows Mobile je Android namenjen poganjanju več različnih naprav. V času pisanja diplomske naloge so bili na voljo štiri mobilni telefoni, do konca leta pa je načrtovanih okoli deset.

Po dolgem zatišju se je junija 2009 pridružil tudi Palm, eden izmed nekdaj največjih proizvajalec dlančnikov, z napravo Palm Pre. Posebnost je integracija telefona s servisi družbenih omrežij, kot so Facebook, LinkedIn, Tweeter.

Danes se odvija bitka na trgu pametnih telefonov – proizvajalci skušajo z novimi pristopi in funkcijami privabiti čim več uporabnikov ter si zagotoviti večji tržni delež. Med njimi je tudi možnost razvoja novih aplikacij.

2.3. Razvoj aplikacij za pametne telefone

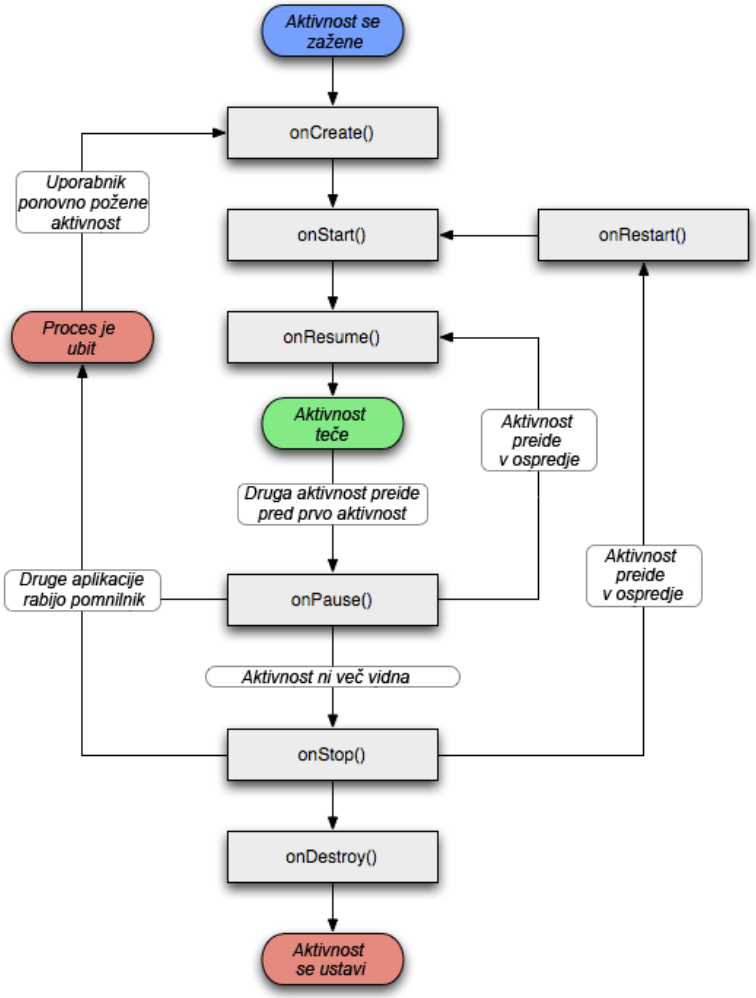
2.3.1. Opredelitev razvoja aplikacij

Vsak, ki bo razvijal aplikacije za mobilne naprave, se mora seznaniti s tremi aspekti razvoja:

- s strojnimi omejitvami naprave
- z življenjskim ciklom aplikacije
- z gradnjo dobrih uporabniških vmesnikov

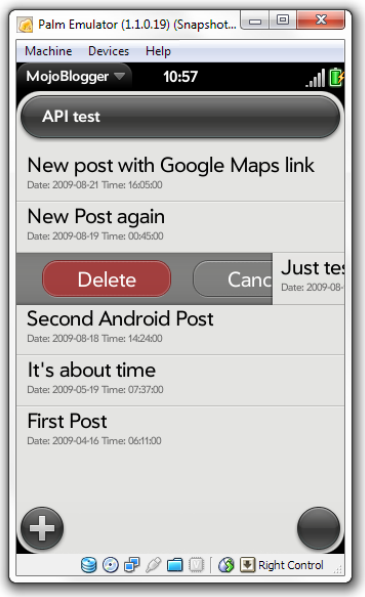
Čeprav so danes pametni telefoni zmogljiv skupek tehnologije, se razvijalci soočajo z omejeno hitrostjo procesiranja, majhnostjo pomnilnika, odvisnostjo od baterijskega napajanja, dragim prenosom podatkov (kadar brezžično omrežje ni na voljo) in velikostjo, s katero je povezan uporabniški vmesnik. Seznanjenost razvijalca z omenjenimi problemi in njihovo obravnavanje je za razvoj in v končni fazi tudi uspeh aplikacije bistvenega pomena. V okviru razvoja aplikacij za mobilne naprave mora razvijalec posvetiti veliko pozornosti učinkoviti kodi, kar vključuje pametno zaseganje in sproščanje pomnilniških in drugih virov, hitre algoritme za procesiranje zahtev, kratke dostope do energijsko potratnih funkcij naprave, čim daljše čase osveževanja ter izogibanje nesmiselnim in potratnim dostopom do omrežja. Optimizacija vseh omenjenih postavk ima pri razvoju za mobilne naprave veliko vlogo.

Za razvoj celovitih aplikacij mora biti razvijalec pozoren tudi na življenjski cikel aplikacij (Slika 3). Ta se sicer razlikuje od operacijskega sistema in implementacije, vendar je pri vseh potrebno aplikacijo na nek način inicializirati, zajeti vire, aplikacijo izvesti, obravnavati izjemne primere (kot so klic, izpad iz omrežja, prazna baterija med delovanjem aplikacije), vzpostaviti predhodna stanja, ko se vračamo iz izjemnih primerov ter aplikacijo končati, ko jo ne potrebujemo več. Večinoma za vse to poskrbi operacijski sistem, razvijalec pa je v okviru življenjskega cikla zadolžen za shranjevanje podatkov ob primernih trenutkih in obveščanje uporabnika o stanju.



Slika 3: Življenjski cikel aktivnosti za Android platformo

Naslednji aspekt kateremu je danes potrebno izkazati vse več pozornosti so uporabniški vmesniki. Ti morajo biti intuitivni, pregledni in vizualno privlačni. Ker je vse več pametnih telefonov z na dotik občutljivim zaslonom in podporo kretnjam je potrebno vmesnike za najboljšo izkušnjo temu tudi prilagoditi. Primer dobro zasnovanega uporabniškega vmesnika je WebOS za Palm Pre, kjer elemente v seznamu brišemo, tako da jih s prstom preprosto odvlečemo iz seznama. Ko element izbrišemo se na njegovem mestu pojavi izbira za potrditev ali preklic (Slika 4). Mobilne naprave so majhne, zato moramo paziti na pravilno razporeditev, prikaz in velikost elementov.



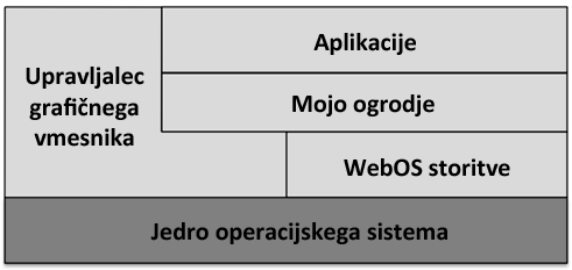
Slika 4: Brisanje elementa iz seznama na Palm Pre telefonu

Ni težko prispeti do zaključka, da je za razvoj aplikacij za pametne telefone potrebna velika mera programerske discipline, izkušenj in kreativnosti. Za razvoj obstaja ogromno razvojnih orodij, ki nam olajšajo programiranje, prav tako pa obstaja nekaj načinov razvoja aplikacij, ki si jih bomo ogledali v nadaljevanju.

2.3.2. Načini razvoja aplikacij

2.3.2.1. Domorodno *(angl. Native)* programiranje

Operacijski sistemi so razdeljeni na več stopenj med katerimi so aplikativni nivo, knjižnice, storitve, gonilniki in jedro sistema. Aplikacije se izvršujejo na aplikativnem nivoju, znotraj navideznega stroja, ki skrbi za upravljanje s pomnilnikom in pravilno izvajanje aplikacije.



Shema 1: Prikaz arhitekturne zgradbe WebOS operacijskega sistema

Ločiti je potrebno:

Domorodno programiranje

Gre za programiranje v programskem jeziku operacijskega sistema, kjer imamo neposreden dostop do strojne opreme in operacijskega sistema. Izvrševanje takih aplikacij je najhitrejše, vendar je ta način zamuden in zahteva veliko mero programerske discipline. Programske vmesnike, knjižnice in funkcije, ki se morajo izvesti hitro je priporočljivo pisati na tak način. Domorodno programiranje dovoljujeta odprto kodna sistema Android z NDK *(angl. Native Development Kit)* in kmalu SymbianOS, medtem ko ostali proizvajalci razvijalcem nudijo samo ogrodja za razvoj aplikacij na aplikativnem nivoju.

Domorodne aplikacije

To so aplikacije, ki se izvajajo na aplikativnem nivoju operacijskega sistema in so pred izvajanjem že prevedene v domorodno kodo. Izvrševanje lahko upravlja izvajalno okolje, neke vrste navidezni stroj, ki skrbi za pravilno izvajanje, varnost in upravljanje s pomnilnikom. Za razliko od aplikacij napisanih v JavaME, Adobe Flash Lite imajo domorodne aplikacije neomejen dostop do vseh funkcionalnosti naprave.

Za razvoj takih aplikacij imamo na voljo programske vmesnike, s katerimi dostopamo do temeljnih funkcij sistema in naprave, preko vodil *(angl. handler)* in metod.

Pri domorodnem programiranju imamo poln dostop do funkcij naprave in operacijskega sistema

2.3.2.2. Java ME

Java Micro Edition je zbirka Javanskih programskih vmesnikov ter komponent, namenjenih razvoju aplikacij za mobilne naprave. Za telefone obstaja konfiguracija CLDC *(angl. Connected Limited Device Configuration)* in specifikacija MIDP *(angl. Mobile Information Device Profile*), ki določata kateri programski vmesniki so vključeni v Java ME.

Izvorna koda aplikacij se pred izvajanjem prevede v vmesno kodo *(angl. byte code*), ta pa se izvede v javanskem navideznem stroju, ki je ločen od operacijskega sistema naprave. To je ena izmed prednosti in omogoča boljšo združljivost med različnimi platformami. Med ostale prednosti spadajo samodejno sproščanje pomnilnika, cena razvoja - vsa orodja so brezplačna in dobra podpora s strani proizvajalcev (Nokia, BlackBerry). Slabosti so strma krivulja priučitve, počasen razvoj ter počasno izvajanje aplikacij.

V prihodnosti jo bo nadomestila JavaFX, ki je skupna platforma za razvoj aplikacij za namizne računalnike in prenosne naprave. Zastavljena je tako, da aplikacije različnih platform delijo večino programske kode, grafičnih elementov in ostale vsebine. Vsebuje skupne programske vmesnike in vmesnike specifične za platformo.

2.3.2.3. .NET Compact Framework

.NET CF aplikacije se podobno kot Java ME aplikacije pred izvajanjem prevedejo v skupni vmesni jezik CIL *(angl. Common Intermediate Language*) in tečejo v skupnem izvajalnem okolju CLR *(angl. Common Language Runtime*). .NET CF je podmnožica polnega .NET ogrodja in poleg knjižnic polnega .NET nabora vsebuje tudi knjižnice napisane specifično za mobilne naprave. Trenutno je na voljo samo na napravah z Windows Mobile operacijskim sistemom, bistvena lastnost pa je razvoj v C# ali VB .NET programskem jeziku. Microsoft takemu načinu pravi upravljana koda *(angl. Managed Code)* in je priporočljiv način razvoja za nove razvijalce, saj je razvoj hitrejši od domorodnega in ni potrebno skrbeti za upravljanje s pomnilnikom. Po drugi strani, pa je izvajanje takih aplikacij počasnejše.

2.3.2.4. Adobe Flash Lite

Adobe Flash Lite je predvajalnik multimedijskih vsebin in aplikacij napisanih v Adobe Flash orodju. Razvoj se loči od ostalih okolij, saj je primarno namenjen animaciji, vendar s skriptnim jezikom ActionScript 3.0 lahko pišemo kompleksne aplikacije, ki omogočajo dostop do datotečnega sistema, uporabljajo brezžično povezavo ter lokacijske in ostale senzorje naprave. Kljub temu je resnih aplikacij napisanih v Adobe Flash Lite zelo malo, saj trenutno Flash vsebino predvajajo samo naprave podjetij Nokia in Sony Ericsson.

Flash Lite ima nekaj ključnih prednosti pred ostalimi. Razvoj je izredno hiter in zato primeren za prototipiranje in testiranje uporabniških vmesnikov - ti so pri napravah z na dotik občutljivim zaslonom zelo pomembni. Aplikacije so kompatibilne z vsemi napravami in

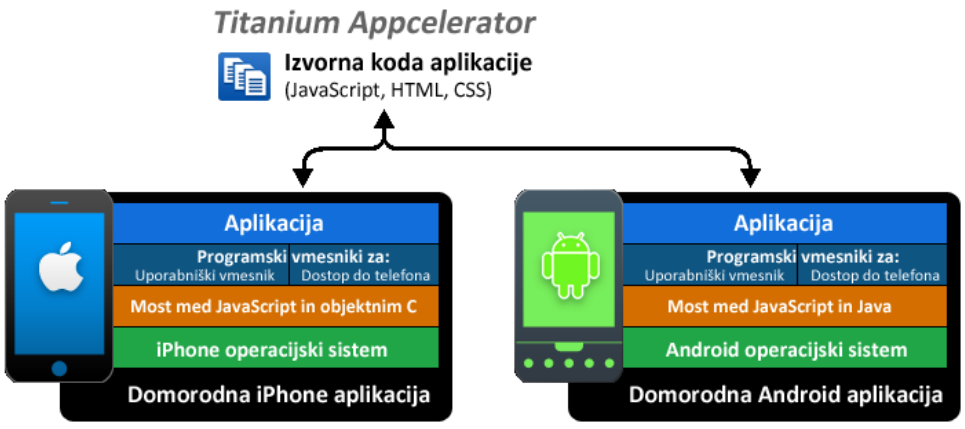
platformami, ki podpirajo Flash Lite – »Write once, run anywhere«. Razvojno okolje za namizne in mobilne aplikacije je enako, prav tako pa je potrebno zelo malo dela, da namizne aplikacije prilagodimo mobilnim napravam in obratno. Slabost Adobe Flash Lite je v slabi podpori proizvajalcev mobilnih naprav, kar pomeni malo končnih uporabnikov, oziroma potencialnih kupcev ter drago razvojno okolje.

2.3.2.5. Druga razvojna ogrodja

Pojavljati so se začela ogrodja, ki omogočajo hitrejši in prijaznejši razvoj domorodnih aplikacij (Motorola MOTODEV) ali pa nudijo možnost izvoza aplikacije, ki smo jo napisali v orodju, za različne naprave (PhoneGap, Appcelerator Titanium). Razvija se v dostopnih programskih jezikih, tehnologijah ter »primi in spusti« načinih izdelavanja uporabniških vmesnikov. Prav tako vsebujejo delčke kode *(angl. snippets)*, ki jih enostavno vgradimo v aplikacijo. Delujejo tako, da razvojno kodo integrirajo z domorodnimi programskimi vmesniki.

Appcelerator Titanium

Je samostojno ogrodje za razvoj aplikacij za platformi iPhone in Android. Aplikacije pišemo v kombinaciji HTML, CSS in JavaScript označevalnih in skriptnih jezikov. Ogrodje Titanium vsebuje programske vmesnike, ki poskrbijo za preslikavo in uporabo domorodnih elementov operacijskega sistema (podatkovna baza, dostop do omrežja, elementi uporabniškega vmesnika, itd) ter most iz JavaScript v objektni C oziroma Javo, ki poskrbi da se koda prevede v domorodno kodo naprave. Prednost pisanja v Appcelerator Titaniumu je hkratni razvoj aplikacije za obe platformi, slabost pa je v tem, da so na voljo samo programski vmesniki, ki nam jih ponuja ogrodje.



Shema 2: Ogrodje Appcelerator Titanium

PhoneGap

Deluje na podoben princip kot Appcelerator Titanium, s tem da poleg iPhone in Android podpira tudi BlackBerry naprave.

Motorola MOTODEV

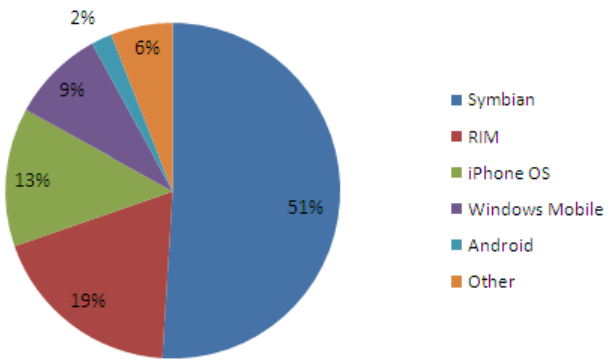
Je razširjeno ogrodje, namenjeno hitremu razvoju aplikacij *(angl. RAD - Rapid Application Development*) za Android platformo, ki ga ponuja proizvajalec mobilnih naprav Motorola. Vsebuje vsa potrebna orodja (Eclipse z dodatki), predloge in izvlečke kode najbolj uporabljenih funkcionalnosti, orodja za podpisovanje, testiranje in izdajo aplikacij.

Kot smo spoznali, lahko aplikacije razvijamo na različne načine, vsak pa ima svoje prednosti in slabosti. Nekateri operacijski sistemi nudijo več kot en način razvoja aplikacij, od razvijalčevih izkušenj in osebnih preferenc pa je odvisno kakšen pristop bo izbral.

2.4. Operacijski sistemi za pametne telefone

Podobno kot operacijski sistemi za namizne računalnike, imajo operacijski sistemi na pametnih telefonih nalogo upravljanja z mobilno napravo in skrbeti za interakcijo z uporabnikom. Danes je v uporabi šest med seboj popolnoma različnih operacijskih sistemov. Vsi omogočajo razvijanje uporabniških aplikacij, vendar se med seboj razlikujejo po načinih razvoja, življenjskih ciklih aplikacij, programskih vmesnikih in funkcionalnostih.

Prikaz tržnega deleža operacijskih sistemov za mobilne naprave v drugem četrtletju leta 2009:



Slika 5: Prikaz tržnega deleža OS za mobilne naprave (Gartner, avgust 2009)

Operacijski sistemi na mobilnih napravah skrbijo za:

- deljenje podatkov znotraj aplikacije in med aplikacijami
- optimalno dodeljevanje procesorskih in pomnilniških virov aplikacijam
- večopravilno izvajanje
- shranjevanje podatkov
- življenjske cikle
- zajemanje uporabniških zahtev

V nadaljevanju si oglejmo kratko zgodovino operacijskih sistemov, kako poteka razvoj aplikacij za njih in kaj jih čaka v prihodnosti.

2.4.1. Symbian OS (Nokia)

Gre za deloma odprt operacijski sistem, ki vsebuje knjižnice, osnovni uporabniški vmesnik in referenčna orodja za delo z mobilno napravo. Razvili so ga v partnerskem podjetju Symbian Ltd., ustanovljenem junija leta 1998, da bi povezal podjetja Nokia, Ericsson, Motorola in Psion. Deset let kasneje je Nokia odkupila preostali delež podjetja in nastala je neprofitna organizacija Symbian Foundation, s člani kot so Nokia, Motorola, Sony Ericsson, NTT DoCoMo,Texas Instruments, Vodafone, Samsung, LG in AT&T. Namen je nadzorovati in razvijati Symbian OS kot popolnoma odprto kodno platformo, pod EPL *(angl. Eclipse Public Licence)* licenco – prehod na odprtokodnost je v teku in bo v celoti končan leta 2010.

Symbian OS omogoča izvajanje programov, nadzoruje strojno opremo in nudi programske vmesnike za dostop do komunikacijskih, multimedijskih, sistemskih in ostalih storitev. Uporabniški vmesnik je prepuščen v implementacijo zunanjim izvajalcem, saj Symbian vsebuje le osnovne razrede in strukturo. Trenutno najbolj znani uporabniški vmesniki so:

- platforma S60 - poganja Nokiine in Samsungove telefone s Symbian sistemom
- UIQ *(angl. User Interface Quartz)* - prilagojen za Sony Ericsson pametne telefone
- MOAP *(angl. Mobile Oriented Applicaion Platform)* - vmesnik prilagojen japonskim mobilnim napravam in uporabnikom DoCoMo mobilnega operaterja

Danes platformo S60, ki teče na Symbian OS 9.4 uporablja večina Nokiinih (med novejšimi sta 5800 in N97) ter nekateri Samsung-ovi (i8910 OmniaHD) pametni telefoni. Peta izdaja (S60v5) je prilagojena napravam z na dotik občutljivim zaslonom ter podpira višje ločljivosti, vendar ne zaznava interakcij z več prsti.

Razvoj za Symbian OS

Domorodne aplikacije za Symbian OS razvijamo v programskem jeziku C++, vendar lahko za večino naprav, ki jih poganja Symbian razvijamo tudi v jezikih kot so Python, Pearl, Ruby, Adobe Flash Lite, .NET C# (uporabljeno je.NET CF); vendar končni uporabnik za poganjanje takih aplikacij na napravi potrebuje tudi orodja, ki omogočajo njihovo izvajanje.

Uradno razvojno okolje predstavlja Carbide.c++, ki temelji na razvojnemu orodju Eclipse z dodatki za podporo Symbian razvoju v C++ programskem jeziku. Komplet za razvoj programov vsebuje potrebna orodja za pisanje, razhroščevanje in testiranje aplikacij na posnemovalniku *(angl. emulator)* in telefonu. Preden dokončano aplikacijo prenesemo na telefon oziroma ponudimo drugim uporabnikom, jo je potrebno digitalno podpisati. V primeru, da aplikacija uporablja zahtevnejše funkcionalnosti naprave (dostop do uporabnikovih podatkov, omrežja, funkcij ki spreminjajo obnašanje telefona) ali pa jo bomo uporabljali v komercialne namene, jo je potrebno podpisati preko programa "Symbian Signed". Zaradi letnega pavšala 200$ in plačila 20$ za vsako novo aplikacijo je sistem nezanimiv za neodvisne in amaterske razvijalce ter nastajajoča podjetja. Aplikacije, ki uporabljajo osnovne zmožnosti, lahko podpiše razvijalec sam.

Poleg razvoja v že omenjenih programskih jezikih, je možno razvijati Java ME in Adobe Flash Lite aplikacije. Prve ustvarjamo s standardnimi orodji za razvoj Java aplikacij na mobilnih napravah (NetBeans, Eclipse), ter potrebujemo orodje za ravzvoj *(angl. Java Development Kit)* za Java ME. V okolju Symbian, podobno kot pri namiznih operacijskih sistemih, tečejo v navideznih strojih, zato so take aplikacije praviloma počasnejše. Symbian OS je eden redkih mobilnih operacijskih sistemov, ki podpira Flash Lite aplikacije.

Trenutno ima razvijanje za Symbian OS v C++ programskem jeziku zaradi specifičnih programskih tehnik in arhitekture strmo krivuljo priučitve. Možnost programiranja v različnih programskih jezikih ga naredi bolj dostopnega za razvijalce, ki so učinkoviti v modernejših jezikih in nimajo želje po priučitvi programiranja v C++. Že sedaj nudi veliko bazo uporabnikov, saj je imela Nokia novembra 2008 47.1% tržni delež na področju pametnih telefonov. Velik problem je bil neobstoj enotnega distribucijskega kanala, kjer bi razvijalci nudili oziroma tržili svoje aplikacije. To so popravili s portalom Ovi (www.ovi.com) maja 2009, ki poleg aplikacij (Ovi Store) nudi tudi ostale storitve kot so elektronsko pošto, multimedijsko vsebino in 10GB oddaljenega diskovnega prostora. Razvijalci ne potrebujejo pristojbine, če želijo aplikacijo ponuditi na Ovi Store, prav tako je brezplačna tudi oddaja aplikacij. Od vsake plačljive aplikacije prejmejo razvijalci 70% njene vrednosti.

Prihodnost Symbian OS

Cilj organizacije Symbian Foundation je odprto kodni operacijski sistem imenovan Symbian^2, ki ga bodo uporabljale vse nove naprave, združil pa bo prednosti platforme S60, UIQ, MOAP in inovacije, ki jih bodo prispevali razvijalci. V bližnji prihodnosti načrtujejo izdelavo Symbian^3 ter Symbian^4 operacijskih sistemov ter poenostaviti potek podpisovanja aplikacij "Symbian Signed".

2.4.2. Windows Mobile (Microsoft)

Windows CE je obsežen operacijski sistem za mobilne, minimalistične naprave ter vgrajene sisteme in temelji na prilagojenih Windows Win32 programskih vmesnikih. Zasnovan je tako, da v osnovi spominja na operacijski sistem Windows za namizne računalnike, vendar ga je po potrebi možno prilagoditi in uporabiti le določene sistemske komponente, funkcionalnosti ter osnovne aplikacije, ki so napravi potrebne. Windows Mobile je tako platforma, katere jedro bazira na operacijskem sistemu Windows CE in je namenjena širokemu naboru mobilnih naprav od pametnih telefonov, dlančnikov, prenosnih predvajalnikov multimedijskih vsebin, do vgrajenih računalnikov v nekaterih avtomobilih (platforma Microsoft Auto; proizvajalci avtomobilov: Hyundai-Kia, Fiat, Ford).

Tržni delež Windows Mobile je od leta 2004 padel za 10% in je konec leta 2008 predstavljal 14% operacijskih sistemov v mobilnih telefonih.

Razvoj za Windows Mobile

Domorodne aplikacije za Mobile platformo razvijamo v Visual C++ programskem jeziku lahko pa razvijamo v jeziku, ki ga podpira .NET Compact Framework (C#, VB.NET), v tako imenovanem načinu z upravljano kodo. Za izvajanje takih aplikacij skrbi izvajalno okolje .NET in pred zagonom aplikacije prevede kodo v domorodno izvajalno kodo. Prednost pisanja programov v upravljanih jezikih, je v tem da ni potrebno skrbeti za pomnilnik, varnost in brisanje objektov, ko jih ne potrebujemo več. Vendar se aplikacije napisane v domorodnem načinu izvajajo hitreje in imajo popoln dostop do strojne opreme naprave. Izbira je na strani razvijalca in je odvisna od njegovih programerskih izkušenj v določenem jeziku ter potrebnih zmogljivosti aplikacije.

Uradno razvojno orodje je Visual Studio .NET in omogoča pisanje, razhroščevanje in testiranje aplikacij, tako na posnemovalniku kot na mobilni napravi.

Na vseh napravah z Windows Mobile sta privzeto naložena privilegirani in normalni certifikat, zato je končno aplikacijo priporočljivo digitalno podpisati, oziroma obvezno v primeru da uporabljamo privilegirane programske vmesnike. Na voljo so trije načini avtentikacije: podpis za privilegirano izvajanje, podpis za normalno izvajanje in nepodpisano.

Stopnja avtentikacije, ki je potreben za izvajanje aplikacije, je odvisen od uporabljenih programskih vmesnikov. Razlikovati moramo avtentikacijo od dovoljenj - ta so odvisna od varnostnih nastavitev v sami napravi. Tako imamo lahko nepodpisano aplikacijo, ki dostopa do privilegiranih funkcionalnosti in če ji to dovolimo, se bo izvedla. Certifikat za podpisovanje aplikacij dobimo preko Microsoftove naveze Mobile2Market, v kateri sta trenutno dva ponudnika digitalnih podpisov: VeriSing in GeoTrust. Certifikat za aplikacije z normalnim in privilegiranim dostopom stane 350$ s katerim lahko podpišemo deset aplikacij.

Aplikacije bodo razvijalci tržili preko distribucijskega sistema »Windows tržnica za mobilne naprave« *(angl. Windows Marketplace for Mobile)*, ki bo na voljo proti koncu leta 2009. Letni pavšal za registracijo v sistem bo znašal 99$ in bo vključeval pravice za izdajo petih aplikacij; za vsako naslednjo bo potrebno odšteti 99$. Študenti, ki so člani programa "DreamSpark" (https://www.dreamspark.com/) bodo aplikacije dodajali brezplačno, vseeno pa bo pristojbina znašala 99$. Od vsake aplikacije bodo razvijalci dobili 70% vrednosti, kupci pa bodo imeli pravico povrnitve stroškov v 24 urah, če z aplikacijo ne bodo zadovoljni.

Prihodnost

Vse verzije Windows Mobile, do vključno zadnje najavljene 6.5, temeljijo na Windows CE operacijskem sistemu in so prilagojene za dlančnike in mobilne naprave, vendar trenutno brez naprednejših funkcionalnosti senzorjev ter na dotik občutljivega zaslona, kot so kretnje in interakcija z več prsti, kar ga je postavilo za konkurenčne operacijske sisteme in naprave. Vse to bo podpiral Windows Mobile 7, ki je načrtovan za leto 2010 in bo napisan popolnoma na novo.

2.4.3. BlackBerry OS (Research In Motion)

Podjetje RIM je s svojimi BlackBerry napravami primarno ciljalo na trg poslovnih uporabnikov. Prvi so leta 2002 ponudili storitev potisne pošte, ki elektronsko pošto potisne na napravo v trenutku ko prispe v naš poštni predal. Poleg tega omogoča popolno sinhronizacijo z Lotus Notes ter Exchange elektronsko pošto, koledarjem, opravki, kontakti in beležko. RIM je novembra 2008 napravami BlackBerry predstavljal 20% trga pametnih telefonov.

Razvoj za BlackBerry

BlackBerry operacijski sistem in priložene aplikacije so napisane v Java programskem jeziku. BlackBerry nudi tri različne pristope k razvoju aplikacij:

- razvoj spletnih aplikacij
- razvoj v "MDS Runtime" (Mobile Data System)
- razvoj Javanskih aplikacij

Načini se razlikujejo po uporabljeni programski sintaksi, enostavnosti kodiranja in stopnjo dostopa do funkcionalnosti naprave. Razvoj spletnih aplikacij izkorišča enostavnost HTML, XHTML, XML in JavaScript tehnologij. Izkoristimo lahko prednosti potisne komunikacije in koncepta "Offline From Queuing", ki zahteve shrani v čakalno vrsto dokler internetna povezava ni zopet vzpostavljena. Ker take aplikacije tečejo na BlackBerry Enterprise strežnikih, do njih pa dostopamo preko brskalnika, ni potrebe po namestitvi aplikacij na napravi. Slabost je v tem, da imajo spletne aplikacije omejeno funkcionalnost. Zato vse zapletene aplikacije razvijamo v Java ME jeziku, za katere so na voljo standardni ter specifično za BlackBerry telefone razviti programski vmesniki. Razvoj v Java ME je najbolj fleksibilen, vendar tudi kompleksnejši od razvoja spletnih aplikacij. Vmesni člen med razvojnima skrajnostnima je "MDS Runtime" razvoj, ki temelji na SOA principih. S pomočjo spletnih storitev lahko zelo hitro razvijemo kompleksno aplikacijo, hkrati pa nudi večjo fleksibilnost od razvoja spletnih aplikacij.

Končne aplikacije lahko ponudimo, oziroma tržimo preko distribucijske aplikacije BlackBerry App World. Pristojbina je enkratna in znaša 200$ ter vključuje pravice za oddajo desetih aplikacij. Za vsakih nadaljnjih deset je zopet potrebno plačati 200$. Aplikacijo je potrebno poslati podjetju RIM v odobritev in jo po končanem postopku prenesti na App World portal.

Prihodnost

RIM si je z napravami BlackBerry ustvaril svojo skupnost uporabnikov in razvijalcev. Uporabnikom nudijo robusten in hiter sistem, z ogromno naprednimi funkcijami, ki so že privzeto vsebovane v napravi. V času pisanja diplomskega dela niso najavili nobenih načrtov za prihodnost. Zaradi neatraktivne cenovne politike in vpričo ogromno konkurenčnih izdelkov, bodo težko pridobili (pozornost) nove razvijalce in nastajajoča podjetja.

2.4.4. WebOS (Palm)

WebOS je najmlajši operacijski sistem za mobilne naprave. Razvilo ga je podjetje Palm in ga izdalo junija 2009. Glavna prednost WebOS je popolna integracija sistema z danes aktualnimi družabnimi omrežji ter razvoj, ki bazira na Web 2.0 tehnologijah, kot so HTML5, CSS in JavaScript. WebOS je v arhitekturnem smislu vdelan operacijski sistem, ki temelji na Linux jedru in vključuje prilagojen uporabniški vmesnik na osnovi spletnih aplikacij. Tudi vse aplikacije napisane za WebOS, so v osnovi spletne aplikacije in jih kot take tudi razvijamo.

Razvoj za WebOS

Aplikacije za WebOS razvijamo s kombinacijo HTML *(angl. Hyper Text Markup Language*) ter CSS *(angl. Cascade Style Sheet*) označevalnih jezikov za prikaz in JavaScript programskega jezika za pisanje izvršilne kode. JavaScript ogrodje imenovano Mojo ponuja

razvijalcem vse potrebne programske vmesnike za razvoj aplikacij za WebOS. Ti nudijo dostop do naloženih aplikacij, njihovih podatkov, vgrajenih storitev, s katerimi dostopamo do funkcionalnosti naprave ter gradnike za izdelavo uporabniških vmesnikov. Struktura aplikacij temelji na "Model-View-Controller" arhitekturnem stilu, za boljše ločevanje podatkov, prikaza in logike. Aplikacije lahko razvijamo v Eclipse orodju, za katerega obstaja zmogljiv dodatek, ki nudi celotno integracijo MojoSDK s samim orodjem.

Končno aplikacijo je potrebno digitalno podpisati. Palmovi strokovnjaki preverijo ali aplikacija ustreza vsem pravilom (varnost, primerna vsebina, ...). Ko je aplikacija potrjena in podpisana, je edini način, da jo ponudimo uporabnikom preko "Palm App Catalog"-a. V času pisanja diplomskega dela ni bil določen pavšal za razvijalce.

Prihodnost

Trenutno je razvojno okolje in orodje v začetni fazi. Na voljo še ni vseh programskih vmesnikov, zato je razvoj aplikacij omejen na enostavne aplikacije, ki temeljijo na uporabi seznama z malo procesiranja. V prihodnosti se bo to spremenilo in pričakujem, da se bo ogromno razvijalcev odločilo za WebOS zaradi enostavnega načina razvoja aplikacij.

2.4.5. iPhone OS (Apple)

iPhone OS je operacijski sistem podjetja Apple in poganja finančno najuspešnejšo napravo na področju pametnih telefonov. Prvi iPhone je izšel junija 2007, do danes pa je bil deležen dveh izboljšav. Vzporedno s strojnimi lastnostmi so dopolnjevali tudi operacijski sistem, tako da je na voljo že tretja verzija. Od začetka prodaje, pa do avgusta 2009 so prodali 21 milijonov naprav iPhone. Apple je novembra 2008 z iPhone OS lasti okoli 11% tržnega deleža pametnih telefonov. Po samo devetih mesecih obstoja distribucijskega portala App Store so slavili prenos milijardte aplikacije, julija 2009 pa je vseboval 65 tisoč aplikacij. Takšni pozitivni podatki o uspešnosti iPhone-a privabljajo mlade razvijalce in nastajajoča podjetja. Prav tako pa je uspeh Apple-a na področju mobilnih naprav povod, da so v igro vstopila tudi druga podjetja z bolj ali manj konkurenčnimi izdelki.

Razvoj za iPhone OS

Aplikacije za iPhone OS razvijamo v objektnem C *(angl. Objective-C)* programskem jeziku, ki temelji na Cocoa aplikacijskem ogrodju. Cocoa lahko primerjamo z .NET ogrodjem v Windows operacijskih sistemih in nudi knjižnice, programske vmesnike, izvajalno ter razvojno okolje. Objektni C je objektno orientiran jezik, ki izhaja iz C++ in vsebuje nekaj korenitih sprememb.

Uradno razvojno orodje je XCode in skupaj z iPhone SDK *(angl. Software Development Kit*) nudi vse potrebno za ustvarjanje, razhroščevanje in testiranje aplikacij na posnemovalniku.

Poganjanje in beta testiranje na napravi je mogoče preko Ad-hoc distribucije. Na ta način lahko razvijalec pooblasti do 100 iPhone telefonov, ki bodo beta aplikacije lahko naložili brez uporabe iTunes Store distribucijskega sistema. V vseh ostalih primerih moramo aplikacije pridobiti preko iTunes, bodisi namizne bodisi mobilne verzije. Da bi aplikacije ponudili ostalim uporabnikom, se je potrebno registrirati kot razvijalec in plačati enkratni pavšal v vrednosti 99$, s katerim dobimo pravice za distribucijo neomejenega števila aplikacij. Aplikacijo lahko ponudimo na iTunes Store portalu brezplačno ali pa določimo ceno, od katere za vsako prodano dobimo 70%. Težava je v tem, da Apple zavrne vsakršno aplikacijo, ki se ne drži strogo definiranih pravil, ima podobno funkcionalnost kot že kakšna izmed obstoječih aplikacij, ali pa škodi Applovemu interesu.

Prihodnost

Apple ima ogromno zvestih uporabnikov, vendar se za razvijalce položaj slabša. Trenutno je na iTunes Store portalu več kot 65 tisoč aplikacij in slogan »There's an app for just about everything«, da obstaja aplikacija za skoraj vse, je kvečjemu trn v peti razvijalcem. Temu kljubuje dejstvo, da Apple z vsako posodobljeno verzijo operacijskega sistema doda nove funkcionalnosti in s tem poveča možnosti za razvoj dodatnih aplikacij. Kot primer bi omenil digitalni kompas v najnovejšem iPhone 3GS, ki bo odprl vrata novim kreativnim razvijalcem.

2.4.6. Android (Google)

Android je odprto kodna platforma za razvoj aplikacij na mobilnih napravah. Izdelal jo je Google skupaj z zvezo Open Handset Alliance, katere člani so v svetu vodilna podjetja na področju mobilnih tehnologij ter programske in strojne opreme. Prednost Android-a oziroma njegove odprtosti je v tem, da imajo tako proizvajalci mobilnih naprav kot razvijalci in uporabniki vpogled v kodo jedra operacijskega sistema, knjižnic in aplikacij. Tudi sami lahko prispevajo k razvoju operacijskega sistema. Bistvena lastnost je v tem, da so prednaložene aplikacije enakovredne aplikacijam, ki jih napišejo razvijalci, saj so uporabljena enaka orodja. Še več, prednaložene aplikacije, kot so telefonski imenik, aplikacije za klicanje in pošiljanje kratkih sporočil, itd., lahko nadomestimo s svojimi. Na tak način lahko proizvajalec ali celo operater, ki napravo ponuja le-to prilagodi po svojih potrebah.

Prednost pred ostalimi operacijskimi sistemi je integracija z Google-ovimi spletnimi storitvami, kot so Gmail, Calendar, Google Maps, GTalk. Novembra 2008 je izšel prvi telefon, ki ga je poganjal Android in sicer HTC G1. Danes pa že poganja preproste prenosnike, oziroma mrežnike *(angl. netbook)*, MID-e *(angl. Mobile Internet Device)*, predvajalnike glasbe.

Razvoj za Android

Aplikacije pišemo v Javi, ki je v tem trenutku najbolj razširjen programski jezik [10], na voljo pa imamo vse programske vmesnike za popoln dostop do naprave. Večina vmesnikov je iz standardne Java 5.0 zbirke, za namizne računalnike, dodali pa so nekatere, ki so napisani specifično za Android-a. Uradno razvojno orodje je Eclipse z dodatki, ki olajšajo pisanje programov, razhroščevanje, testiranje in izdelavo uporabniških vmesnikov. Pri razvoju lahko aplikacijo zaženemo in testiramo v posnemovalniku, ali pa neposredno na telefonu. Aplikacije tečejo vsaka v svojem navideznem stroju, imenovanem Dalvik. Ta skrbi za pravilno izvajanje, varnost in upravljanje s pomnilnikom, Android pa poskrbi za hkratno izvajanje večih aplikacij in pravilno razporejanje virov, kadar kakšna izmed aplikacij ni več potrebna, ali pa so viri naprave potrebni drugi aplikaciji.

Google ponuja tudi orodja za domorodno programiranje na stopnji operacijskega sistema. Kodira se v C/C++ programskem jeziku. Prednosti so hitrejše izvajanje, direkten dostop do strojne opreme in manjše zaseganje virov. Slabosti pisanja v domorodni kodi so zapleten razvoj, zmanjšana kompatibilnost med napravami, ni dostopa do programskih vmesnikov. Ni namenjeno razvoju končnih aplikacij, ampak izdelavi procesorsko zahtevnih operacij, lastnih knjižnic, programskih vmesnikov in operacij, ki se morajo izvesti hitro.

Preden končano aplikacijo namestimo na telefon ali ponudimo ostalim uporabnikom, jo je potrebno digitalno podpisati. Za vse Android aplikacije je dovolj in celo tipično da so samo-podpisane *(angl. Self-signed)*, postopek pa se izvede s čarovnikom znotraj Eclipse orodja. Tako podpisano aplikacijo, lahko naložimo na telefon ali pošljemo na Android Market. Če želimo aplikacijo ponuditi na distribucijskem kanalu, se moramo prijaviti kot razvijalec in plačati enkratno pristojbino 25$.

Prihodnost

Čeprav Android platforma še ni tako uspešna kot iPhone, gre za operacijski sistem, ki se bo zagotovo obdržal. Proizvajalci mobilnih telefonov so Android zelo dobro sprejeli. V tem trenutku so na trgu 4 naprave, do konca leta 2009 pa napovedujejo diverzifikacijo naprav in povečanje pošiljke za 900% v primerjavi z začetkom leta 2009 [8]. S tem bi Android pridobil vodilno vlogo na področju mobilnih operacijskih sistemov. Po drugi strani za razvijalce zgodba ni tako bleščeča. Eden izmed večjih razvijalcev aplikacij za Android platformo, Larva Labs, je spisal članek [9] o težavah pri distribuciji svojih aplikacij in zaključil, da se v tem trenutku prodaja aplikacij ne more primerjati z iPhone-om. Glavni razlog tiči v slabo zasnovani distribucijski aplikaciji Android Market, kot slabost pa navajajo obstoj samo enega načina plačevanja (preko Google Checkout), omejen opis aplikacije s 325 besedami brez zaslonskega posnetka, osredotočenost na brezplačne aplikacije (prikaz plačljivih je prikrit v meniju) in končne uporabnike, ki so vajeni brezplačnih aplikacij. Google sicer obljublja

nadgraditev Android Marketa in možnost plačevanja na več načinov, vendar je vse več uporabnikov mnenja, da so svojo možnost za prevzem vodstva na področju mobilnih naprav že zamudili.

	iPhoneApp Store	**Android Market**	**BlackBerry App World**	**Windows Mobile Marketplace**	**Palm App Catalog**	**Nokia Ovi Store**
Ekskluzivni vir aplikacij	da	ne	ne	ne	da	ne
Plačljive / zastonj aplikacije	da / da	da / da	da / da	da / da	da / da	da / da
Odjamalec na telefonu	da	da	da	da	da	da
Odjemalec na računalniku	da	ne	ne	ni info.	ni info.	da
Podpora ostali vsebini (slike, melodije, ...)	ne	ne	ne	ne	ne	da
Način plačila	iTunes + kreditna kartica	Google Checkout + kreditna kartica	PayPal	kreditna kartica, operater	ni info.	kreditna kartica,operater
Povračilo kupnine	ne	24 ur	ne	24 ur	ni info.	ne
Vsebina trgovinevezana na operaterja	ne	da	ne	da	ni info.	da
Delež, ki ga prejme razvijalec	70%	70%	80%	70%	ni info.	70%
Prisojbina za razvijalce	Enkratna, 99$	Enkratna, 25$	Enkratna, 200$	Letna, 99$	ni info.	Brezplačno
Število aplikacij	Neskončno	Neskončno	200$ za 10 aplikacij	5 zastonj, vsaka nadaljna 99$	ni info.	Neskončno
Minimalna cena plačljivih aplikacij	0.99$	0.99$	2.99$	ni info.	ni info.	ni info.

Tabela 1: Primerjalna tabela distribucijskih orodij (vir: Gizmodo.com)

	Apple iPhone 3GS	HTC Hero	Palm Pre	BlackBerry Storm	Nokia N97
Zaslon	3.5", 320x480, kapacitivni	3.2", 320x480, kapacitivni	3.1", 320x480, kapacitivni	3.25", 320x480, "SurePress"	3.5", 360x640, odporni
Večtočkovni dotik	da	da	da	ne	ne
Tipkovnica	ne	ne	da	ne	da
Pomnilnik	16GB / 32 GB	512MB	8GB	1GB	32GB
Razširljiv pomnilnik	ne	priložena 2GB (največ 16GB)	ne	priložena 8GB	največ 16GB
Fotoaparat / ostrenje / fleš	3MP / da / ne	5MP / da / ne	3.2MP / da / da	3.2MP / da / da	5MP / da /da
Snemanje	da	da	da	da	da
GPS	da	da	da	da	da
Wi-Fi	da	da	da	ne	da
3D pospeševanje	da (OpenGL ES)	da	ne	ne	da (OpenGL ES)
teža	135g	135g	135g	155g	150g

Tabela 2: Primerjalna tabela strojnih lastnosti pametnih telefonov (vir: Gizmodo.com)

3. Predstavitev aplikacije

3.1. Splošno o aplikaciji

V tem poglavju bom predstavil praktični del diplomskega dela, ki ga predstavlja aplikacija za tri popolnoma različne operacijske sisteme: Android, WebOS in iPhone OS. Aplikacija je napisana z namenom, da se seznanim s koncepti, principi ter zahtevnostjo razvoja za posamezno platformo.

Osrednja funkcionalnost je branje in dodajanje prispevkov na spletni dnevnik, ki temelji na Googlovi storitvi Blogger. Uporabnik lahko s telefonom, na katerem je nameščena aplikacija bere in dodaja prispevke na lastni spletni dnevnik kjerkoli in kadarkoli. Uporablja brezžično *(angl. WiFi oziroma Wireless)* povezavo ali 3G omrežje. Uporabnik ima možnost določiti svojo GPS lokacijo in jo vključiti v vsebino prispevka. Lokacija bo na spletnem dnevniku vidna kot povezava na storitev Google Maps.

Slika 6: Primer treh variant aplikacije razvite za tri različne telefone

Aplikacija uporabniku omogoča sledeče funkcionalnosti:

- branje spletnih dnevnikov
- dodajanje in spreminjanje prispevkov
- določanje GPS lokacije
- shranjevanje nastavitev

Aplikacija pokriva naslednje funkcionalnosti naprave in operacijskega sistema:

- povezava na splet preko brezžične povezave ali 3G omrežja
- dostop do spletne storitve (Google Blogger)
- prevzem in obdelava podatkov v XML notaciji, v obliko ki je primernejša za obdelavo
- pošiljanje podatkov na spletni strežnik
- dostop do GPS modula
- uporaba mehanizma za shranjevanje nastavitev aplikacije

3.2. Zasnova in uporaba aplikacije

Aplikacija je sestavljena iz petih sklopov:

3.2.1. Seznam vseh prispevkov spletnega dnevnika

Gre za prikaz prispevkov, ki smo jih sprejeli s strežnika v XML notaciji in preoblikovali v bolj berljivo obliko. Na seznamu prikažemo naslove ter datume in čase objave prispevkov. V nastavitvah aplikacije lahko uporabnik nastavi ali bo prikazan tudi predogled vsebine prispevkov ter koliko znakov bo prikazalo. Ta pogled vsebuje tudi navigacijo do pogleda za dodajanje novih prispevkov in nastavitev aplikacije, sam prikaz navigacijskih elementov pa se razlikuje med operacijskimi sistemi. Uporabniški vmesnik je prilagojen interakciji s prsti, tako da z dotikom na prispevek zaženemo pogled za branje prispevkov. Android zazna tudi daljše dotike (1 sekundo), s katerim prikažemo pogled za spreminjanje izbranega prispevka.

3.2.2. Pogled za branje prispevkov

V tem pogledu prikažemo podrobnosti o posameznem prispevku. Sam sem se odločil prikazati naslov, vsebino in kategorije, v katere spada prispevek. Vsebina, ki jo sprejmemo, je oblikovana s HTML elementi in če platforma omogoča, prikažemo oblikovano vsebino. Trenutno samo Android podpira prikaz oblikovane HTML vsebine iz lokalnega niza, pri ostalih je torej prikazana v izvirni, neoblikovani obliki. Pogled vsebuje navigacijski element, ki zažene pogled za spreminjanje že vnesenih prispevkov.

3.2.3. Pogled za spreminjanje prispevkov

Pogled vsebuje tri polja, ki jih napolnimo s podatki o prispevku, katerega ga želimo spremeniti. Spodaj je gumb za potrditev spremembe in preklic. Pri Android-u imamo namenski fizični gumb za nazaj, tako da ga v uporabniškem vmesniku ne potrebujemo.

3.2.4. Pogled za dodajanje novih prispevkov

Vsebuje tri polja, ki predstavljajo naslov, vsebino in kategorije prispevka ter preklopni gumb za omogočanje zajeme GPS lokacije in prikaz v vsebini prispevka. V polje kategorije lahko vnesemo več kategorij, ki morajo biti ločene z vejico. Uporabnik ima možnost prispevek poslati ali začasno shraniti, v primeru, da ga bo kasneje dopolnil ali pa se v tistem trenutku nahaja nekje kjer ni omrežja. Tak prispevek se je na seznamu vseh prispevkov prikazan z oznako »Osnutek«.

3.2.5. Nastavitve aplikacije

V nastavitvah hranimo podatke o nazivu spletnega dnevnika ter uporabniškem računu. Iz naziva sestavimo URL naslov dnevnika, ki ga potrebujemo za branje in pošiljanje zahtev na strežnik. Podatke o uporabniškem računu (elektronski naslov in geslo) potrebujemo za prijavo, če želimo dodajati in spreminjati prispevke. S preklopnim gumbom izbiramo ali se bo prikazal predogled vsebine prispevka, v vnosnem polju pa določimo koliko znakov bomo prikazali v predogledu.

3.3. Primeri uporabe

3.3.1. Branje prispevkov spletnega dnevnika

Primer uporabe omogoča uporabniku vpogled v seznam prispevkov spletnega dnevnika. Z dotikom na posamezen prispevek se v pogledu prirejenemu za branje prikažejo podrobnosti.

Predpogoj

Telefon mora imeti omogočen dostop do interneta.

Osnovni tok – opisuje normalno delovanje aplikacije, kadar želi uporabnik brati prispevke

1. Uporabnik zažene aplikacijo
2. Samodejno se prenesejo vsi prispevki s strežnika na telefon
3. Izpiše se urejen seznam prispevkov – novejši so na vrhu
4. Uporabnik z dotikom zaslona izbere prispevek, ki ga podrobneje zanima
5. Zažene se pogled za prebiranje posameznega prispevka s podrobnostmi, kot so naslov, vsebina, datum in čas ter kategorije

Alternativni tok – opisuje delovanje aplikacije, ob prvem zagonu

1. Uporabnik prvič zažene aplikacijo
2. Aplikacija uporabnika opozori, da naslov spletnega dnevnika ni določen
3. Zažene se pogled z uporabniškimi nastavitvami
4. Uporabnik vnese naslov spletnega dnevnika
    - Če bo prispevke dodajal in spreminjal vnese še uporabniško ime in geslo
5. Uporabnik pritisne gumb »Shrani« oziroma »Nazaj«
6. Nastavitve se samodejno shranijo
7. Aplikacija nadaljuje z delovanjem od koraka 2, pri osnovnem toku

3.3.2. Vnos novega prispevka v spletni dnevnik

Primer uporabe omogoča uporabniku vnos naslova, vsebine in večih kategorij objave. Dodatna možnost je vključitev GPS lokacije, ki jo aplikacija doda kot povezavo na storitev Google Maps v vsebini prispevka. Uporabnik lahko uporablja HTML notacijo za oblikovanje vsebine.

Predpogoj

Telefon mora imeti omogočen dostop do interneta. Uporabnik mora imeti račun za spletni dnevnik za storitev Google Blogger.

Osnovni tok – opisuje normalno izvajanje, kadar so na voljo vsi potrebni podatki

1. Uporabnik iz menija v izbere »Dodaj prispevek« *(angl. Add post)*
2. Aplikacija v ozadju preveri ali je uporabnik v nastavitvah določil uporabniško ime ter geslo
3. Uporabnik izpolni obrazec (naslov, vsebina, kategorije) za dodajanje prispevka in pritisne gumb »Objavi« *(angl. Publish)*
4. Aplikacija se prijavi v Blogger spletno storitev
5. Aplikacija prenese podatke na strežnik
6. Prenos je uspešen, aplikacija obvesti uporabnika in se vrne na seznam vseh prispevkov

Alternativni tok 1 – kadar uporabnik pomotoma vnese napačne uporabniške podatke

1. Uporabnik vnese podatke o prispevku in pritisne gumb »Objavi«
2. Aplikacija se skuša prijaviti v sistem Blogger, vendar avtentikacija ni uspešna

3. Aplikacija podatke o vnosu začasno shrani in odpre nastavitve, kjer lahko uporabnik posodobi svoje podatke
4. Uporabnik pritisne gumb »Nazaj«
5. Aplikacija predpostavi, da so podatki spremenjeni in pravilni in se ponovno skuša prijaviti
6. Če avtentikacija zopet ni uspešna se vrne na korak 3, drugače pa nadaljuje s korakom 5 pri osnovnem toku.

Alternativi tok 2 – uporabnik prispevek napiše, vendar ga shrani za kasnejši prenos, ko bo v dosegu brezžičnega omrežja

1. Uporabnik vnese podatke o prispevku in pritisne gumb »Shrani osnutek« *(angl. Save draft)*
2. Aplikacija podatke o prispevku shrani in se vrne na seznam vseh prispevkov
3. Shranjena objava se prikaže na vrhu seznama in je označena z »Osnutek« *(angl. Draft)*
4. Ko uporabnik želi poslati prispevek, ga z dotikom izbere
5. Naloži se pogled za urejanje prispevka, kjer ga lahko dopolni ali objavi
6. Uporabnik pritisne na gumb »Objavi« *(angl. Publish)*
7. Aplikacija nadaljuje delovanje s korakom 5 pri osnovnem toku

3.3.3. Urejanje prispevka

Primer uporabe uporabniku omogoča spreminjanje naslova, vsebine in kategorij prispevka.

Predpogoj

Telefon mora imeti omogočen dostop do interneta. Uporabnik mora lastiti račun za spletni dnevnik za storitev Google Blogger. Prispevek mora biti objavljen na spletnem dnevniku.

Osnovni tok

1. Uporabnik s seznama prispevkov z dotikom izbere prispevek, ki bi ga rad uredil
    a. Android ima možnost daljšega dotika (1 sekunda), ki takoj prikaže pogled za urejanje prispevka
2. Zažene se pogled za branje prispevka
3. Iz menija izbere možnost »Uredi prispevek« *(angl. Edit post)*
4. Zažene se pogled za urejanje prispevka
5. Uporabnik lahko spremeni naslov, vsebino in kategorijo prispevka in ko konča pritisne gumb »Posodobi« *(angl. Update)*

6. Aplikacija se prijavi v sistem Blogger in posodobi prispevek

3.4. Uporabljene storitve

Aplikacija uporablja Google-ovo storitev za gostovanje spletnih dnevnikov Blogger. V okviru te storitve je razvijalcem na voljo vrsta programskih vmesnikov za večino današnjih programskih jezikov (Java, JavaScript, .NET, PHP, Python). Vmesniki omogočajo enostaven dostop do spletnih dnevnikov, avtorizacijo uporabnikov ter dodajanje, urejanje in komentiranje posameznih prispevkov.

Pri izdelavni nisem uporabljal obstoječih programskih vmesnikov, temveč sem potrebne funkcionalnosti sprogramiral sam. Tako sem imel boljši pregled nad napravami in samim razvojem, kar je tudi bistvo diplomskega dela. Pomagal sem si z dobro dokumentiranim protokolom storitve Blogger.

3.4.1. Prijava v sistem

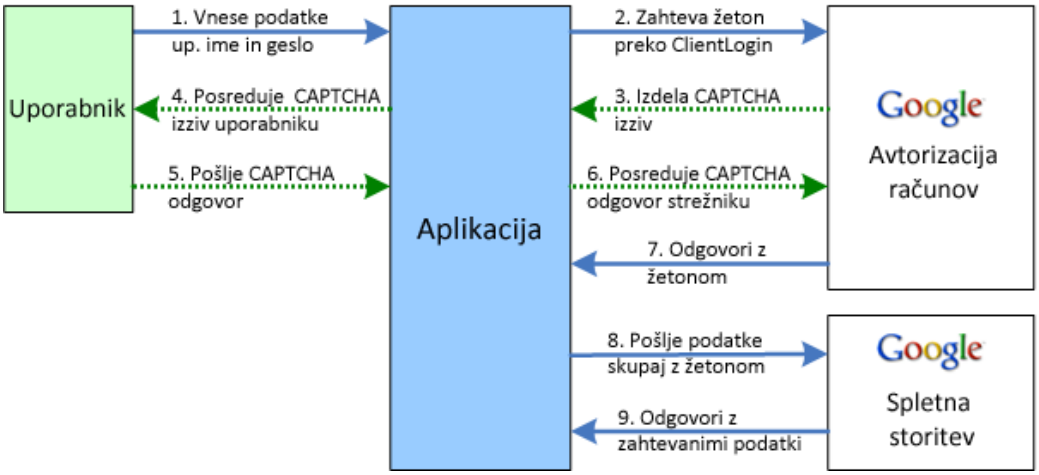
V sistem se prijavimo tako, da pošljemo `HTTP-POST` zahtevo z množico parametrov (elektronska pošta, geslo, storitev, tip računa in identifikacija aplikacije) na naslov:

```
https://www.google.com/accounts/ClientLogin
```

V primeru da sta uporabniško ime in geslo napačna, strežnik odgovori s statusno kodo `403 Forbidden`. Potrebno jo je obravnavati in obvestiti uporabnika. Kadar je prijava uspešna se strežnik odzove s statusno kodo `200 OK` in avtorizacijskim žetonom v vsebini odziva. Pridobljeni žeton je potrebno implementirati v glavo *(angl. header)* zahteve v vseh naslednjih interakcijah znotraj iste seje, kjer je potrebna prijava (dodajanje, spreminjanje).

Na shemi je prikazan potek interakcije uporabnika z aplikacijo in spletno storitvijo. Če uporabnik večkrat vpiše napačno geslo, strežnik pošlje CAPTCHA izziv. Prikazan je s črtkano črto. Ker odklepanje računa v takem primeru ni možno znotraj aplikacije, je to potrebno storiti ročno na naslovu:

```
https://www.google.com/accounts/DisplayUnlockCaptcha?service=blogger
```



Shema 3: Interakcija med uporabnikom, aplikacijo in spletno storitvijo

3.4.2. Branje spletnega dnevnika

Kadar želimo dostopati do vseh prispevkov v spletnem dnevniku pošljemo `HTTP-GET` zahtevo na naslov:

```
http://<ime_spletnega_dnevnika>.blogspot.com/feeds/posts/default
```

Za takšno poizvedbo, oziroma za branje se ni potrebno prijaviti. Strežnik se odzove s statusno kodo `200 OK` in seznamom prispevkov v Atom 1.0 standardu XML notacije v telesu odziva.

Poleg Atom 1.0 standarda Blogger ponuja tudi prevzem seznama prispevkov v JSON *(angl. JavaScript Object Notation)* notaciji, ki jo s pridom lahko uporabimo v JavaScript programskem jeziku. Prednost JSON notacije je v tem, da so podatki predstavljeni v obliki seznamov in elementov, do katerih enostavno dostopamo z načinom, ki mu pravimo notacija s piko *(angl. dot notation ali dot syntax)*. Če želimo, da nam Blogger pošlje seznam v JSON obliki je potrebno spletnemu naslovu dodati parameter `?alt=json`.

Primer JSON notacije:

```
"entry":[{
      "id": {
            "$t":"postID"
      },
      "published": {
            "$t": "2009-08-21T16:05:00.001-07:00"
      },
      "updated": {
            "$t": "2009-08-21T16:05:09.890-07:00"
      },
      "category": [{
            "scheme": "http://www.blogger.com/atom/ns#",
            "term": "testing"}
      ],
```

```
      "title": {
            "type": "text",
            "$t": "New post with Google Maps link"
       },
      "content": {
            "type": "html",
            "$t": "This post was writen on Android telephone."
      },
      "link": [{
            "rel": "replies",
            "type": "application/atom+xml",
            "href": "http://blogID.blogspot.com/ ",
            "title": "Post Comments"
      }],
      "author": [{
            "name": {
                  "$t": "Sašo"
            },
            "email": {
                  "$t": "noreply@blogger.com"
            }
      }]
}]
```

Primer dostopa do avtorja članka z notacijo s piko:

```
      var authorName = entry.author.name.$t;
```

Nadaljnja obdelava in prikaz sprejete vsebine je odvisna od platforme, zato jo bom opisal v naslednjem poglavju.

3.4.3. Dodajanje prispevkov

Zajete podatke o prispevku, je potrebno pred pošiljanjem preoblikovati v XML obliko po Atom 1.0 standardu s korenskim elementom `<entry>`:

```
<entry xmlns='http://www.w3.org/2005/Atom'>
  <title type='text'>Naslov prispevka<title>
  <content type='xhtml'>
    <div xmlns="http://www.w3.org/1999/xhtml">
      <p>Besedilo prispevka v HTML obliki</p>
    </div>
  </content>
  <category scheme="http://www.blogger.com/atom/ns#" term="blog" />
  <category scheme="http://www.blogger.com/atom/ns#" term="razvoj" />
</entry>
```

Tako predstavljene podatke pošljemo v telesu `HTTP-POST` zahtevka. V glavi zahtevka določimo, da gre za `application/atom+xml` tip vsebine *(angl.* `ContentType`*)*.

Pred tem se je potrebno prijaviti in pridobljen avtorizacijski žeton dodati v glavo zahtevka. V primeru, da smo se znotraj iste seje že prijavili, uporabimo stari avtorizacijski žeton:

```
Authorization: GoogleLogin auth=<naš_avtorizacijski_žeton>
```

Storitev Blogger na strežniku ustvari prispevek z vsebino, ki smo jo poslali in vrne statusno kodo `201 CREATED` ter kopijo prispevka z dodanimi elementi (id, čas objave, ..) v vsebini odziva.

3.4.4. Spreminjanje prispevkov

Pri spreminjanju moramo biti pozorni na to, da upoštevamo prispevek z vsemi XML elementi, čeprav omogočamo samo spreminjanje naslova, vsebine in kategorij. Po zajemanju podatkov iz obrazcev posodobimo elemente `<title>`, `<content>` in `<category>` v XML predstavitvi prispevka:

```
<entry gd:etag='W/"CUYDSXo8fSp7ImA9WB9UFkU."'>
      <id>tag:blogger.com,1999:blog-blogID.post-postID</id>
      <published>2006-11-08T18:10:00.000-08:00</published>
      <updated>2006-11-08T18:10:14.954-08:00</updated>
      <title type='text'>Nov naslov</title>
      <content type='html'>Posodobljena vsebina prispevka</content>
      <link rel='alternate' type='text/html'
            href='http://blogName.blogspot.com/2006/11/nov-naslov.html'>
      </link>
      <link rel='self' type='application/atom+xml'
            href='http://blogName.blogspot.com/feeds/posts/default/postID'>
      </link>
      <link rel='edit' type='application/atom+xml'
            href='http://www.blogger.com/feeds/blogID/posts/default/ >
      </link>
      <category scheme="http://www.blogger.com/atom/ns#" term="blog" />
      <category scheme="http://www.blogger.com/atom/ns#" term="razvoj" />
      <author>
            <name>Sašo</name>
            <email>noreply@blogger.com</email>
            <uri>http://www.blogger.com/profile/profileID</uri>
       </author>
</entry>
```

Tako oblikovan prispevek pošljemo v telesu `HTTP-PUT` zahteve na naslov:

```
http://<spletni_dnevnik>.blogspot.com/feeds/posts/default/postID
```

3.5. Izboljšave aplikacije

Aplikacijo sem izdelal za potrebe diplomskega dela in je za platformo Android popolnoma delujoča. Za telefona Palm Pre in iPhone je implementirano branje spletnega dnevnika ter shranjevanje uporabniških nastavitev.

Funkcionalnost aplikacije lahko izboljšamo z uporabo sledečih storitev:

Storitev Picasa za nalaganje slik

Uporabniku omogočimo izbiro slik, ki jih je posnel z vgrajeno kamero in prikaz v prispevku. Ker storitev Blogger ne podpira nalaganja slik bi uporabili storitev Picasa in jih vključili v vsebino prispevka.

Storitev Youtube za nalaganje video vsebin

Podobno kot pri slikah, bi lahko uporabili storitev za nalaganje video vsebin, ki jih je uporabnik posnel z vgrajeno kamero. Uporabili bi storitev Youtube in jih vključili v vsebino prispevka.

Uporabniško izkušnjo lahko izboljšamo z naslednjimi implementacijami:

Uporaba profilov

Aplikacija trenutno omogoča branje in spreminjanje enega spletnega dnevnika. S profili bi lahko uporabnik dodal informacijo o več spletnih dnevnikih. Na zaslonu bi prikazovali prispevke bodisi mešano ali pa organizirano v skupine po nazivu spletnega dnevnika. Uporabnik bi lahko nastavil katere spletne dnevnike naj prikaže v seznamu.

Branje različnih tipov spletnih dnevnikov

Aplikacija je prirejena za branje spletnih dnevnikov, ki temeljijo na storitvi Blogger. Ker zna prispevke izluščevati iz spletnih virov, ki temeljijo na Atom 1.0 standard, lahko prikazuje tudi spletne dnevnike drugih ponudnikov. Lahko bi jo izboljšali tako, da prispevke izlušči iz spletnih virov, ki temeljijo na drugih standardih – recimo Wordpress-ov XML.

Nepovezan dostop

Aplikacija trenutno ne shranjuje celotnega spletnega dnevnika zato podatki, ko aplikacijo zapremo niso več na voljo. Ob vsakem zagonu se s spleta prenese vsebina dnevnika. Aplikacijo bi lahko razvili tako, da shrani podatke o spletnem dnevniku v bazo, tako da ob naslednjem zagonu nebi dostopali do spletnega dnevnika. Storitev Blogger celo podpira poizvedbo, kjer kot parameter pošljemo identifikacijsko število zadnjega prispevka, ki je

shranjen v naši bazi. V primeru, da je zadnji tudi v spletni bazi, ne pošlje ničesar. Na tak način lahko tudi zmanjšamo prenos podatkov preko omrežja.

Prikaz komentarjev v pogledu za branje prispevka

Aplikacija trenutno ne obravnava komentarjev. Lahko bi implementirali kodo, ki bi v seznamu prispevkov prikazala število komentarjev. V podrobnejšem pogledu za branje pa tudi vsebino in avtorje komentarjev.

4. Razvoj za Android, WebOS in iPhone

V času pisanja diplomskega dela je razvijalcem in uporabnikom na razpolago šest svojevrstnih operacijskih sistemov. Osredotočil sem se na tri, ki imajo po mojem mnenju največji potencial za prihodnost in za njih razvil aplikacijo za branje in urejanje spletnih dnevnikov.

Za Android platformo sem se odločil zato, ker gre za Google-ov operacijski sistem za mobilne naprave. Google-ove spletne storitve so že nekaj časa v samem vrhu, zato me je zanimalo ali je svojo kvaliteto prenesel na domeno mobilnih naprav. WebOS je zaradi razvoja s kombinacijo JavaScript, HTML in CSS zelo zanimiv operacijski sistem, saj se danes s to tehnologijo ukvarja velika večina ljudi. Enostavnost programskega jezika in razširjenost HTML ter CSS notacije, je prednost za Palm in dovolj dober razlog, da vključim omenjeni sistem v svojo diplomsko nalogo. Ni težko opredeliti zakaj sem izbral tudi razvoj za iPhone. Pozornost je pridobil predvsem zaradi številnih zgodb, kjer so razvijalci obogateli čez noč [11]. Po drugi strani v primerjavo nisem vključil razvoja za operacijske sisteme Symbian, BlackBerry in Windows Mobile. Symbian ima največji tržni delež na področju pametnih telefonov, vendar razvoj aplikacij zanj nikoli ni bil pretirano priljubljen. Razvoj za BlackBerry naprave sem izpustil, saj ima neugodne finančne pogoje za amaterske razvijalce. Razvoj za Windows Mobile bi bil sicer zanimiv, vendar so v tem trenutku operacijski sistemi, ki jih bom obravnaval, novejši in zmogljivejši.

4.1. Metodologija dela

Za vsako platformo sem si vzel okoli 20 dni, da sem se seznanil s koncepti izdelave, orodji za izdelavo aplikacij, razvojnim okoljem, programskim jezikom, programskimi vmesniki, in v končni fazi izdelavo aplikacije.

Najbolj sem bil pozoren na razvojno okolje, saj prav ta najbolj vpliva na učinkovitost pisanja aplikacij, če predpostavimo, da se bo razvoja lotil nekdo, ki ima približno enako stopnjo znanja v programskih jezikih.

Vsak izmed sklopov mi je, kot sem pričakoval, povzročal težave. Ti problemi so bili privajanje in težave z razvojnim okoljem, učenje sintakse programskega jezika, težave z orodjem za razhroščevanje, nepregledna dokumentacija itd. Na koncu poglavja bom v opisni obliki podal ugotovitve, do katerih sem prišel s primerjavo razvoja za omenjene operacijske sisteme.

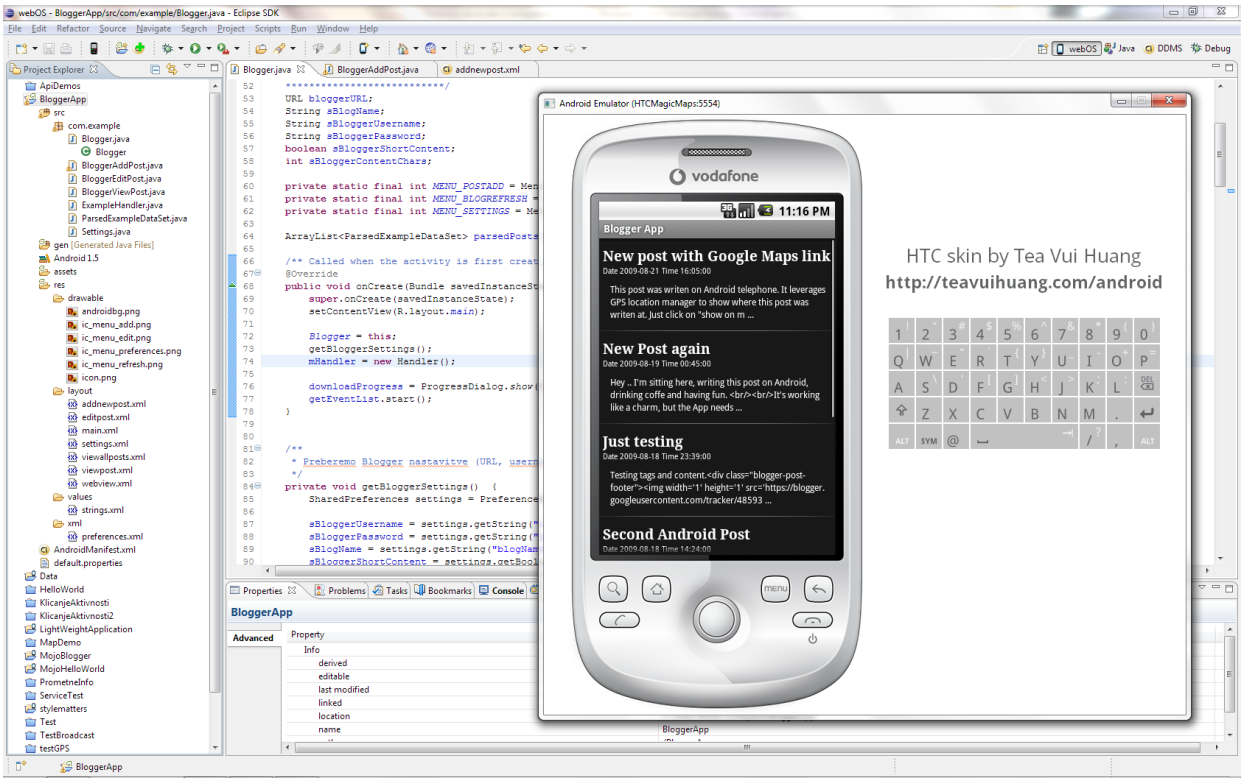
4.2. Razvoj aplikacije za Android

4.2.1. Opis razvojnega okolja in programskega jezika

Ena izmed prednosti razvoja za Android platformo je zagotovo razvojno okolje Eclipse v navezi z dodatkom ADT. Manj izkušenemu razvijalcu nudi pomoč v obliki:

- sprotne detekcije napak v kodi, s predlaganimi rešitvami, ki jih lahko tudi izvede
- seznama možnih metod in tvorjenja nastavkov zanje
- tvorjenja Setter/Getter metod za vmesnike objektov pri enkapsulaciji
- zmogljivega razhroščevalnika DDMS *(angl. Dalvik Debug Monitor Service)*
- dovršitve kode *(angl. code completion)*
- prikaza JavaDoc dokumentacije za izbrano metodo

Okolje z dodatkom nudi zelo dobro integracijo in praktično ni potrebe po drugih orodjih. Ob začetku pisanja nam ponudi čarovnike in tvori nastavke kode, med samim kodiranjem in testiranjem, pa samodejno pregleduje kodo in predlaga rešitve. Orodje za razhroščevanje je zelo zmogljivo in nudi možnost nadzorovanja naprave, simuliranja klicev, sporočil in GPS modula. Ko je aplikacija končana, jo preprosto podpišemo z vgrajenim čarovnikom.



Slika 7: Razvojno okolje Eclipse in posnemovalnik delovanja telefona z Android-om

4.2.2. Struktura map ogrodja

```
[ime projekta]
  src/
    [ime paketa]
      imeAktivnosti.java
  gen/
    [ime paketa]
      R.java
  Android 1.5/
  assets/
  res/
    drawable/
      icon.png
    layout/
      main.xml
    values/
      strings.xml
      colors.xml
    xml/
  AndroidManifest.xml
  default.properties
```

Mapa / datoteka	Opis
/src	V paketu se znotraj mape nahajajo Java datoteke naše aplikacije. Vsaka aktivnost je napisana v svoji datoteki.
R.java	Datoteka kjer so shranjeni kazalci na vire, deklarirani v podmapah mape /res. Tvori jo Eclipse samodejno, preko nje pa se v kodi sklicujemo do virov.
/res	Korenska mapa kjer se nahajajo viri aplikacije.
/drawable	Vsebuje slikovne vire, kot so ikona aplikacije, ikone za meni, ozadja in slike uporabljene v aplikaciji.
/layout	Vsebuje uporabniške vmesnike v XML obliki.
/values/strings.xml	Datoteka s pari ključ-vrednost, ki se uporabljajo pri gradnji uporabniških vmesnikov, predvsem pri lokalizaciji aplikacije.
/xml	Vsebuje razne vire v XML obliki, kot je recimo datoteka za nastavitve aplikacije.
AndroidManifest.xml	Hrani bistvene podatke o aplikaciji, kot so ime aplikacije, komponente aplikacije, dovoljenja za sistemske komponente, knjižnice, itd.
default.properties	Opisuje lastnosti za gradnjo *(angl. build)* projekta.

Tabela 3: Opis map in privzeto tvorjenih datotek za Android platformo

Organizacija kode je pregleda in nisem potreboval veliko časa za privajanje. Razredi so zaključena celota, ki lahko kličejo druge razrede ali uporabljajo vire, ki so nam na voljo preko datoteke R.java. Pograjal bi sistem dovoljenj, saj recimo razvijalca ne opozori, da je implementiral kodo, ki potrebuje dostop do GPS modula, ni pa določil ustreznih dovoljenj. Aplikacija v tem primeru ne deluje pravilno, problem pa ni tako trivialno odkriti.

4.2.3. Implementacija aplikacije

V Androidu vsak pogled predstavimo z aktivnostjo. Ta predstavlja podrazred, ki deduje od `Activity` baznega razreda. Ker imamo v aplikaciji pet različnih sklopov (pregled, dodajanje,

spreminjanje, …), je bilo potrebno implementirati pet različnih aktivnosti, vsako s svojim uporabniškim vmesnikom. Podatke med aktivnostmi pošiljamo preko tako imenovanih namer *(angl. intent)*. Poleg aktivnosti imamo na voljo komponente, kot so storitve, prejemnike objav in ponudnike vsebine. V okviru diplomskega dela teh komponent nisem potreboval, zato jih ne bom podrobneje opisoval. Poglejmo si implementacijo aktivnosti.

4.2.3.1. Pošiljanje podatkov med aktivnostmi

Pošiljanje podatkov v nameri:

Kadar želimo iz ene komponente preiti v drugo bodisi znotraj iste aplikacije bodisi med različnimi aplikaciji moramo uporabiti namero. To je objekt, ki ne izvaja nobene operacije, pač podatkovna struktura v kateri so definirani podatki in operacija, ki se bo izvedla. Svoje podatke dodajamo v namero z ukazom `putExtra`.

```
...

// Inicializacija nove namere. Parametra predstavljata trenutni kontekst in
// aktivnost, kateri bomo namero s podatki poslali
Intent postInfo = new Intent(Blogger, BloggerViewPost.class);

// Določimo dodatne podatke, ki jih bomo poslali v nameri
postInfo.putExtra("title", parsedPostsDataSet.get(pos).getPostTitle());
postInfo.putExtra("content", parsedPostsDataSet.get(pos).getPostContent());
postInfo.putExtra("tags", parsedPostsDataSet.get(pos).getPostTags());

// Zaženemo aktivnost z namero v parametru
startActivity(postInfo);

// Primer klicanja aktivnosti, kadar pričakujemo vrnjen rezultat
// ali pa želimo vedeti kdaj se klicana aktivnost konča.
// Potrebna je metoda onActivityResult(), ki bo rezultate obravnavala
startActivityForResult(new Intent(this, AddPost.class, REQ_CODE_ADD);

...
```

Sprejem podatkov iz namere v drugi aktivnosti:

```
...

// Kopiranje vsebine namere v spremenljivko
Intent postInfo = getIntent();

// dostop do podatka 'title' v nameri in izpis na zaslonu
String postTitle = postInfo.getStringExtra("title");
labelPostTitle.setText(postTitle);

...
```

Namere so zmogljivi mehanizem v platformi Android. V aplikaciji namreč lahko tudi definiramo na kakšne namere se bo odzivala in na tak način recimo zaženemo aplikacijo, ko sprejmemo tekstovno sporočilo. Slabost je v tem, da preko namer ne moremo pošiljati kompleksnih podatkovnih struktur, zato se moramo zateči k uporabi statičnih spremenljivk.

4.2.3.2. Branje in izluščevanje spletnega dnevnika

Za Android obstaja programski vmesnik SAX *(angl. Simple API for XML)* za izluščevanje podatkov v XML obliki. SAX je dogodkovno usmerjen, kar pomeni da sproži metodo (`startDocument()`, `endDocument()`, `startElement()`, `characters()`, `...`), ko pride do določenega elementa. Zanj je potrebno definirati upravljalca *(angl. handler)*, ki bo podatke iz ustreznih XML elementov shranjeval v podatkovno strukturo in spletni naslov, kjer se XML dokument nahaja. Prikazana je samo inicializacija izluščevalnika, za celoten postopek potrebujemo še razred s podatkovno strukturo in razred, kjer je definiran upravljalec.

```
...

// Inicializiramo SAX parser
SAXParserFactory spf = SAXParserFactory.newInstance();
SAXParser sp = spf.newSAXParser();
XMLReader xr = sp.getXMLReader();

// Inicializiramo upravljalca in ga določimo parserju
ExampleHandler myExampleHandler = new ExampleHandler();
xr.setContentHandler(myExampleHandler);

// Sestavimo URL iz naziva spletnega dnevnika, ki smo ga prebrali iz
// nastavitev
URL url = new URL("http://" + sBlogName +
                  ".blogspot.com/feeds/posts/default";


// Zaženemo SAX, kot parameter podamo naslov spletnega dnevnika
xr.parse(new InputSource(new StringReader(url.openStream())));
parsedPostsDataSet = myExampleHandler.getParsedData();

...
```

Implementacija izluščevanja podatkov je zamudno opravilo, kodo pa zelo težko uporabimo na drugače oblikovanih podatkih. Predvsem tukaj bi lahko z boljšimi programskimi vmesniki razvijalcem olajšali in pohitrili delo.

4.2.3.3. Prikaz prispevkov spletnega dnevnika

Obdelane podatke na zaslon prikažemo preko `SimpleAdapter` gradnika za sezname. Problem nastane, ker omenjeni gradnik ne zna brati podatkov iz naših lastnih podatkovnih struktur, ampak samo iz polj. Ker imamo podatke v obliki ključ-vrednost, jih je potrebno preoblikovati in v polje shraniti v `<HashMap<String, String>>` obliki.

Blogger.java

```
...

// Inicializacija polja oblike <HashMap<String, String>>
ArrayList<HashMap<String,String>> list = new
ArrayList<HashMap<String,String>>();
// sprehodimo se skozi vse prispevke v podatkovni strukturi in jih shranimo
// v polje list
for (int i = 0; i < parsedPostsDataSet.size(); i++) {
      item = new HashMap<String,String>();
      item.put("title", parsedPostsDataSet.get(i).getPostTitle());
      item.put("publishedDate", "Date " +
            parsedPostsDataSet.get(i).getPostPublishedDate() +
            " Time " +
            parsedPostsDataSet.get(i).getPostPublishedTime());
      item.put("content", parsedPostsDataSet.get(i).getPostContent());

      list.add(item);
}
// Inicializiramo gradnik s sledečimi parametri
SimpleAdapter notes = new SimpleAdapter (
      this,       // kontekst
      list,       // podatkova struktura, kjer so podatki/prispevki

      // datoteka, kjer je definirana razporeditev elementov
      R.layout.viewallposts,
      // katere vrednosti bomo izpisali
      new String[] {"title","publishedDate", "contentShort"},
      // v katero polje bomo izpisali vrednost
      new int[] { R.id.title, R.id.date, R.id.body});

// prikažemo seznam prispevkov na zaslon
setListAdapter(notes);

...
```

Zgoraj omenjena koda izpiše prispevke z razporedom elementov, ki je opisan v viewallposts.xml datoteki.

viewallposts.xml

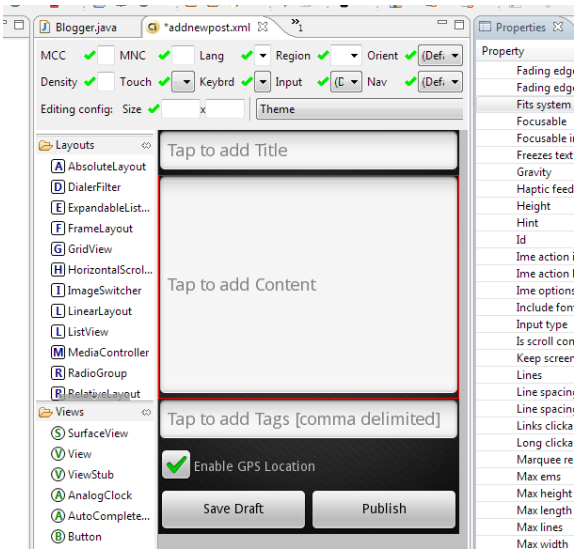
```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android=http://schemas.android.com/apk/res/android
      android:orientation="vertical"
      android:layout_width="fill_parent"
      android:layout_height="wrap_content">

      <!-- TITLE Style -->
      <TextView android:id="@+id/title"
          android:layout_width="fill_parent"
          android:layout_height="wrap_content"
          .../>
```

```
        <!-- DATE Style -->
        <TextView android:id="@+id/date"
            android:layout_width="fill_parent"
              android:layout_height="wrap_content"
              .../>

        <!-- BODY Style -->
        <TextView android:id="@+id/body"
            android:layout_width="fill_parent"
              android:layout_height="fill_parent"
              .../>
</LinearLayout>
```

Prikazano kodo, sem skrajšal na mestih označenih z »...« zaradi boljše preglednosti. Koda, ki sem jo odstranil je bila namenjena oblikovanju izpisa in ni bistvenega pomena. Pomemben je element `android:id="@+id/title`, ki je identifikator polja `TextView`. Eclipse samodejno tvori kazalce na takšne identifikatorje in jih hrani v datoteki R.java. To nam omogoča, da do teh polj znotraj kode dostopamo s kodo:

```
TextView mPostTitle = (TextView) findViewById(R.id.title);
mPostTitle.setText = ("Naslov polja");
```

Uporabniške vmesnike gradimo na »povleci in spusti« način znotraj okolja Eclipse. Pri izdelovanju nimamo velikega nadzora, zato je priporočljivejše pisati v XML kodi. Problem je ker je tak način počasnejši in ni primeren za začetnike. Pričakujem, da se bo ta aspekt orodja izboljšal, saj gre za izredno pomemben del razvoja aplikacij.



Slika 8: Odsek orodja za izdelavo uporabniških vmesnikov znotraj Eclipse-a

4.2.3.4. Upravljanje z nastavitvami aplikacije

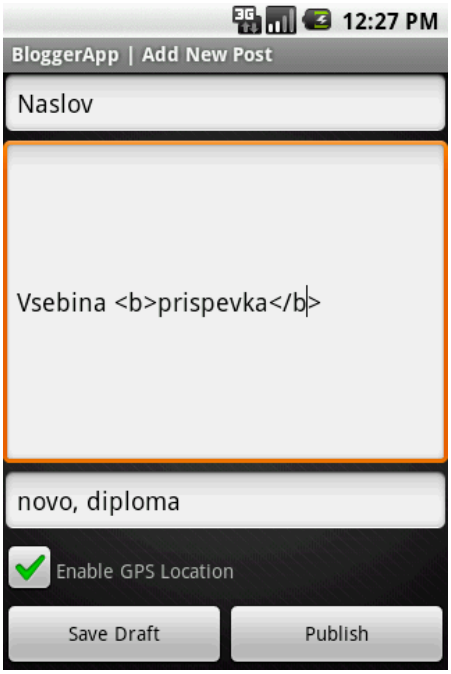
Android nudi enostaven mehanizem za shranjevanje in dostop do nastavitev aplikacije. Razporeditev in tip elementov opišemo v datoteki `/xml/nastavitve.xml`, nato pa v aktivnosti, ki bo zagnala pogled z nastavitvami kličemo funkcijo:

```
...

addPreferencesFromResource(R.xml.nastavitve);

...
```

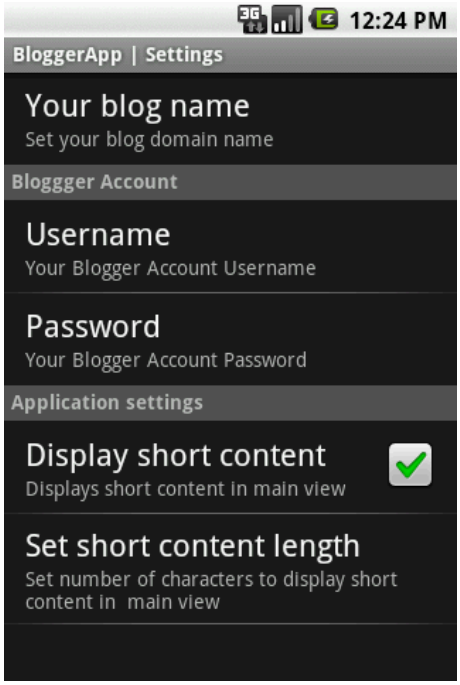
V glavni aktivnosti dostopamo do nastavitev in podatkov v njih s kodo:

```
...

SharedPreferences settings =
      PreferenceManager.getDefaultSharedPreferences(this);

// Preberemo nastavitev z nazivom blogName.
// V primeru da nastavitev ne obstaja vrnemo null
String sBlogName = settings.getString("blogName", null);

...
```

Na tak način lahko hitro realiziramo mehanizem za shranjevanje, vendar motita dve dejstvi. Datoteko z razporeditvijo elementov moramo napisati ročno, poleg tega pa ne obstaja tip nastavitve za hranjenje nizov, pri katerih je vsebina skrita (uporabno predvsem za hranjenje gesel).



Slika 9: Pogled za dodajanje novega prispevka



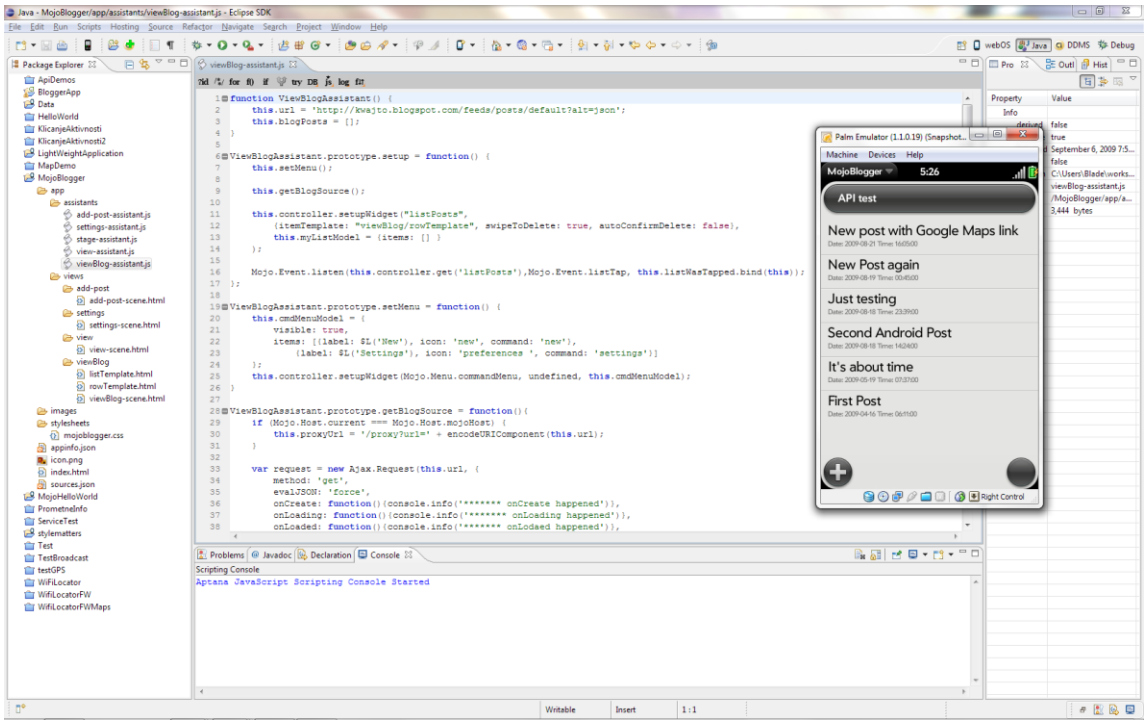
Slika 10: Pogled za shranjevanje nastavitev aplikacije

4.3. Razvoj aplikacije za WebOS

4.3.1. Opis razvojnega okolja

Razvojno okolje je isto kot za Android platformo, vendar je Palm-ov dodatek za Eclipse precej manj zmogljiv. Detekcija napak je šibka in je omejena na označevanje nepravilnega vejanja oklepajev in zaklepajev, nepravilno napisanih rezerviranih besed in funkcij ter označevanje napak v prireditvenih stavkih. JavaScript je ohlapen jezik in spremenljivk ni potrebno inicializirati niti jim določiti tip podatka. To sicer lahko vzamemo kot prednost, v večini primerov pa ima lahko začetnik (in v obsežnih projektih tudi izkušen razvijalec) na račun tega ogromno problemov. Recimo, da določimo `spremenljivka = "Neko besedilo"` in kasneje v kodi prikažemo pogovorno okno `showAlertDialog({message: spremenljivka},...)`, nas prevajalnik ne bo opozoril, da `spremenljivka` še ne obstaja. To je sicer trivialen primer problema, ampak včasih iskanje take (slovnične) napake vzame ogromno časa.

Druga velika težava za začetnike je odsotnost pomoči pri gradnikih in specifičnih funkcijah za WebOS, ki bi nam predlagala katere metode in lastnosti imajo objekti. Dokler razvijalec ne bo dobro poznal gradnikov, bo večino časa iskal njihove lastnosti v dokumentaciji WebOS platforme na Palm-ovi internetni strani.



Slika 11: Razvojno okolje Eclipse s posnemovalnikom delovanja Palm Pre telefona

4.3.2. Struktura map ogrodja

```
[ime projekta]
  /app
    /assistants
      mainApp-assistant.js
  /views
    /mainApp
      mainApp-scene.html
  /images
  /stylesheets
    main.css
appinfo.json
icon.png
index.html
source.json
```

Mapa / datoteka	Opis
/assistants	Vsebuje kodo pomočnikov, ki uporabljajo krmilnike za prizore v JavaScript programskem jeziku. Datoteke so označene s pripono –assistant.js.
/views	Vsebuje s HTML in CSS oblikovane uporabniške vmesnike – prizore. Datoteke so označene s pripono –scene.js in jih sistem avtomatsko povezuje s pomočniki, ki imajo enako ime pred pripono –assistant.js. Datoteke, ki se navezujejo na posamezen prizor organiziramo v podmape.
/images	Korenska mapa kjer se nahajajo slikovni viri aplikacije.
/stylesheets	Vsebuje CSS datoteke s stili za elemente uporabniških vmesnikov.
appinfo.json	Datoteka z osnovnimi podatki o aplikaciji.
icon.png	Ikona aplikacije.
index.html	Prva datoteka, ki se prikaže ko aplikacijo zaženemo. V pomočniku prizorišča lahko potisnemo drug prizor in jo povozimo.
source.json	Datoteka v kateri je določeno katere pomočnike in poglede bomo potrebovali za vsak prizor, prizorišče ali aplikacijo.

Tabela 4: Opis map in privzeto tvorjenih datotek za WebOS platformo

Imel sem težave z organizacijo datotek, še posebej z delitvijo pomočnikov na tiste, ki upravljajo z aplikacijo, prizoriščem in prizorom. Za nekatere operacije je potrebno ustvariti pomočnika za prizorišče, za druge pa lahko dostopamo do nadzornika prizorišča kar preko pomočnika prizora. To so predvsem začetniške težave in menim, da bi z izkušnjami izginile.

4.3.3. Implementacija aplikacije

Aplikacija je predstavljena s prizoriščem *(angl. stage)*, ta pa je sestavljen iz več prizorov *(angl. scenes)*. Naš primer aplikacije vsebuje eno prizorišče; začetni pogled s seznamom, dodajanje, branje in nastavitve pa so prizori. Aplikacija, prizorišče in prizori imajo vsak svojega pomočnika *(angl. assistant)*. Pomočnik prizorišča skrbi za objekte na nivoju aplikacije in potisne prvi prizor:

```
StageAssistant.prototype.setup = function() {
```

```
        this.controller.pushScene('viewBlog');
}
```

Pomočnik prizora skrbi za obravnavanje interakcije uporabnika, dogodkov, prikazuje podatke in potiska nove prizore ko so potrebni. Vsak izmed pomočnikov ima instanco svojega krmilnika *(angl. controller)*.

4.3.3.1. Pošiljanje podatkov med prizori

Kadar hočemo podatek poslati drugemu prizoru, ga vključimo v klic `pushScene()` kot parameter:

viewBlog-assistant.js

```
...
// Ko kliknemo na gumb dostopamo do nadzornika prizorišča in potisnemo
prizor 'editPost' s podatki o prispevku
this.controller.stageController.pushScene('editPost', this.blogData);

...
```

V prizoru `editPost`, podatke preberemo v konstruktorju prizora in jih shranimo v spremenljivko. Če želimo podatke vrniti prizoru, ki je sprožil naš prizor, to naredimo tako, da naš prizor zapremo z ukazom `popScene()` in kot parameter vključimo podatke, ki jih želimo poslati prvemu prizoru. Razlog za to je v tem, da se prizori nalagajo na tako imenovan sklad prizorov in če bi ga ponovno potisnili, bi lahko sklad ciklično rastel v neskončnost.

editBlog-assistant.js

```
// Primer sprejema podatkov v konstruktorju prizora
function EditPostAssistant(argFromPusher) {
        this.currentValue = argFromPusher;
 }

...

// primer zapiranja prizora in vključevanja podatka
this.controller.stageController.popScene(this.responseCode);
```

4.3.3.2. Branje in izluščevanje prispevkov iz spletnega dnevnika

Do spletnega dnevnika na internetu dostopamo z enostavno kodo:

```
...

this.url = 'http://' + blogName +
           '.blogspot.com/feeds/posts/default?alt=json';

var request = new Ajax.Request(this.url, {
```

```
        method: 'get',
        evalJSON: 'force',
        onComplete: this.gotResults.bind(this),
        onFailure: this.failure.bind(this)
});

...
```

Metodi `Ajax.Request` podamo naslov, kjer se nahajajo prispevki in s parametri določimo, da gre za GET zahtevo ter da bomo dobili odgovor v JSON notaciji. Metoda asinhrono prevzame podatke in če je prenos uspešen kliče funkcijo `gotResults(transport)`, kjer bomo podatke shranili v podatkovno strukturo. Omenili smo, da storitev Blogger omogoča prikaz spletnega dnevnika v JSON notaciji. To dejstvo lahko uporabimo pri implementaciji branja podatkov.

```
ViewBlogAssistant.prototype.getBlogSource = function(){
        // Odziv na našo zahtevo je shranjen v spremenljivki transport
        var blogSource = transport.responseJSON;

        // Korak nebi bil potreben, če vrednosti nebi bile shranjene v
        // podseznamu $t. Zato moramo vrenosti shraniti v ustrezek ključ
        for (i = 0; i < blogSource.feed.entry.length; i++) {
                this.blogPosts.push({
                        id: blogSource.feed.entry[i].id.$t,
                        published: blogSource.feed.entry[i].published.$t;,
                        updated: blogSource.feed.entry[i].updated.$t,
                        category: blogSource.feed.entry[i].category,
                        title: blogSource.feed.entry[i].title.$t,
                        content: blogSource.feed.entry[i].content.$t
                });
        }
        // Pridobljene podatke shranimo v spremenljivko myListModel
        this.myListModel.items = this.blogPosts;

        // Gradniku za seznam sporočimo, da so se podatki spremenili
        this.controller.modelChanged(this.myListModel, this);
}
```

Branje spletnega dnevnika z interneta in prevzem podatkov je bil implementiran zelo hitro. Izluščevalnik podatkov ni potreben in je v primeru dobro oblikovanega vira potrebno zelo malo dela. WebOS blesti kar se tiče prikaza podatkov predstavljenih v JSON obliki.

4.3.3.3. Prikaz prispevkov spletnega dnevnika

Razvoj za WebOS upošteva MVC arhitekturni slog, kar pomeni da potrebujemo pogled, kjer definiramo kako bomo prispevke prikazali, kontrolni del, kjer implementiramo logiko potrebno za izpis in podatkovni del, ki predstavlja podatke, ki jih bomo izpisali.

Ko pridobimo podatke o spletnem dnevniku, jih enostavno izpišemo v seznam, ki ga inicializiramo na način prikazan spodaj. Ker so podatki na voljo kasneje, se seznam zažene

brez podatkov. Z ukazom `modelChanged()`, seznamu sporočimo, da smo podatke pridobili, oziroma spremenili in naj osveži vsebino:

```
this.controller.modelChanged(this.myListModel, this);
```

viewBlog-assistant.js - predstavlja kontrolni del in v okviru tega tudi podatkovni model

```
...

// Dostopamo do lastnosti gradnika z identifikatorjem "listPosts"
// S parametrom itemTemplate določimo obliko izpisa vrstice v seznamu
// myListModel so podatki,ki jih bomo izpisali
this.controller.setupWidget("listPosts",
      {itemTemplate: "viewBlog/rowTemplate"},
      this.myListModel = {items: [] }
);

...
```

viewBlog-scene.html – glavni prikaz, kjer vključimo gradnik Mojo.List

```
// predvidimo mesto kjer bomo izpisali naslov spletnega dnevnika
<div id="main" class="palm-hasheader">
      <div class="palm-header left" id="blogTitle"></div>
</div>

// dodamo gradnik List, z identifikatorjem "listPosts"
<div id="listPosts" x-mojo-element="List"></div>
```

rowTemplate.html – tu določimo kako bo oblikovana posamezna vrstica v seznamu

```
<div class="palm-row multi-line" id='#{data}_example' x-Mojo-tap-
highlight="momentary">
      <div class="palm-row-wrapper">
            <div class="postTitle">#{title}</div>
            <div class="date">#{published}</div>
      </div>
</div>
```

Uporabniški vmesnik ročno kodiramo v z uporabo HTML in CSS elementov, zato je gradnja počasna. Palm bo moral ponuditi orodje za izdelavo vmesnikov, če želi pridobiti razvijalce.

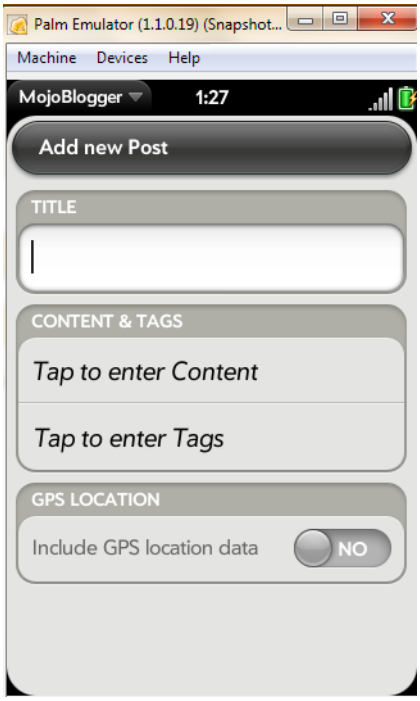
4.3.3.4. Upravljanje z nastavitvami

Za shranjevanje aplikacijskih nastavitev uporabimo programski vmesnik Mojo.Depot. Na ta način lahko v obliki ključ-vrednost shranjujemo kakršne koli JavaScript objekte.

```
...

// Inicializacija baze za shranjevanje nastavitev
```

```
var db = new Mojo.Depot({name: "bloggerDB", replace: false},
                    this.dbOpenOK.bind(this),
                    this.dbOpenError.bind(this),);

...

// v bazo shranimo vrednost iz tekstovnega polja pod ključem 'blogname'
db.add("blogname", fieldBlogName.model.value,
       this.dbAddOK.bind(this),
       this.dbAddError.bind(this));

// branje podatka iz nastavitev
this.blogName = db.get("blogname",
                    this.dbReadOK.bind(this),
                    this.dbReadError.bind(this));
```

Uporabniški vmesnik in inicializacijo elementov na prizoru, je potrebno ustvariti ročno, zato je izdelava mehanizma za shranjevanje nastavitev počasna.



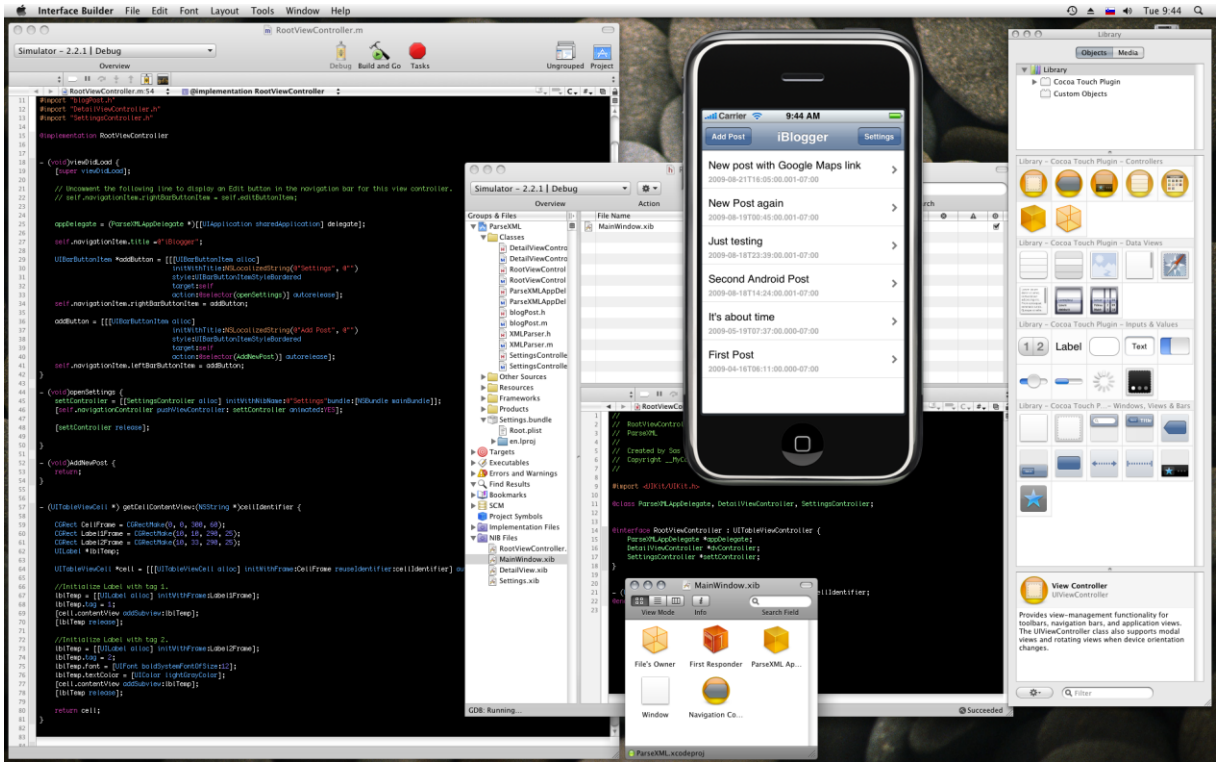
Slika 12: Prizor za dodajanje novega prispevka

Slika 13: Prizor za shranjevanje nastavitev aplikacije

4.4. Razvoj aplikacije za iPhone OS

4.4.1. Opis razvojnega okolja in jezika

Aplikacije za iPhone telefon razvijamo v XCode orodju in je z novo verzijo iPhone OS 3.1 možno samo na MacOS X operacijskih sistemih verzije 10.5.8 ali novejših. Ogrodje je zelo dobro zasnovano in ponuja tudi zmogljivo orodje za gradnjo uporabniških vmesnikov in analizo zmogljivosti aplikacije.



Slika 14: Okolje XCode, posnemovalnik iPhone telefona ter graditelj uporabniškega vmesnika

4.4.2. Struktura map ogrodja

```
[ime projekta]
  /Classes
    imeRazred.h
    imeRazred.m
  /Other Sources
    main.m
  /Resources
    Info.plst
    MainWindow.xib
  /Frameworks
  /Products
  /Settings.bundle
```

Mapa / datoteka	Opis
/Classes	Vsebuje kodo razredov, ki je po načelih objektnega C razdeljena na glavo (datoteke .h) in implementacijo (datoteke .m).
/Other Sources	Koda ostalih datotek, ki so povezani z aplikacijo. Datoteka main.m, ki je privzeto v tej mapi, je potrebna za zagon aplikacije in jo v večini primerov ni potrebno spreminjati.
/Resources	Korenska mapa kjer se nahajajo viri aplikacije, vključno z uporabniškimi vmesniki.
/Resources/Info.plst	Datoteka z osnovnimi podatki o aplikaciji.
/Resources/MainWindows.xib	Privzeta datoteka s samodejno tvorjenim uporabniškim vmesnikom. Vmesnike urejamo v graditelju uporabniških vmesnikov.
/Frameworks	Vsebuje potrebne knjižnice za uporabljena ogrodja.
/Settings.bundle	Paket za shranjevanje nastavitev aplikacije, kjer definiramo elemente

Tabela 5: Opis map in privzeto tvorjenih datotek za iPhone platformo

Koncept ločevanja deklaracije atributov in metod od implementacije je zelo zanimiv, pogrešam pa mehanizem, ki bi za to poskrbel namesto razvijalca.

4.4.3. Implementacija aplikacije

Aplikacijo je bilo potrebno kodirati v objektnem C jeziku in je eden težjih konceptov, ki ga je potrebno proučiti pri razvoju za iPhone. Ni si težko predstavljati, da večina mladih razvijalcev obupa nad razvojem ravno zaradi programskega jezika. Drug zelo pomemben koncept je upravljanje s pomnilnikom. Čeprav Cocoa ogrodje podpira samodejno sproščanje pomnilnika, ta mehanizem ni vključen v samo napravo, zato je za učinkovito delovanje aplikacije zajemanje in sproščanje obveznost razvijalca. Ker razvoj temelji na MVC arhitekturi, je potrebno za vsak pogled določiti tudi krmilnik in znotraj njega podatke. Poleg tega ločujemo vmesnike razreda od implementacije.

4.4.3.1. Pošiljanje podatkov med razredi

Za izmenjavo podatkov med različnimi razredi lahko skrbi predstavnik *(angl. delegate)* ali krmilnik razreda, izbira je odvisna od načina dostopa.

Preko predstavnika

Razred BloggerAppDelegate hrani podatke o prispevkih s spletnega dnevnika, ki smo jih prevzeli s strežnika in izluščili. Razred RootViewController je po drugi zadolžen za prikaz teh podatkov v seznamu. Da bi dostopali do podatkov določimo predstavnika razreda BloggerAppDelegate, preko katerega bomo dostopali do podatkovnih struktur in metod.

RootViewController.hs

```
@class BloggerAppDelegate;

@interface RootViewController : UITableViewController {
```

```
	ParseXMLAppDelegate *appDelegate;
	...
}
```

RootViewController.m

```
// Inicializiramo predstavnika
appDelegate = (BloggerAppDelegate *)
	[[UIApplication sharedApplication] delegate];

...

// Dostop do podatkovne strukture preko predstavnika
NSInteger numOfPosts = [appDelegate.blogPosts count];
```

Neposredno preko krmilnika

Ob dotiku na prispevek v seznamu, se zažene pogled za branje, ki ga krmili razred DetailViewController, kateremu moramo poslati podatek o naslovu in vsebini prispevka. V tem primeru ne moremo iz razreda DetailViewController prebrati podatke o prispevku, ker še ne vemo katerega je izbral uporabnik. Zato je potrebno podatke iz razreda RootViewController poslati razredu DetailViewController. To storimo tako, da inicializiramo krmilnik razreda, kateremu bomo podatke poslali.

```
// Vgrajena metoda, ki se zažene kadar se dotaknemo vrstice v seznamu
- (void)tableView:(UITableView *)tableView
didSelectRowAtIndexPath:(NSIndexPath *)indexPath {

	if (dvController == nil)
		// zasežemo/inicializiramo krmilnik
		dvController = [[DetailViewController alloc]
			initWithNibName:@"DetailView"
			bundle:[NSBundle mainBundle]];

	// Preko predstavnika dostopamo do prispevka
	blogPost *onePost = [appDelegate.blogPosts
				objectAtIndex:indexPath.row];

	// Krmilniku določimo podatek, ki ga bo obravnaval
	dvController.onePost = onePost;

	// Zaženemo krmilnik
	[self.navigationController pushViewController: dvController
				animated:YES];

	// sprostimo vir, ker ga ne potrebujemo več
	[dvController release];
}
```

4.4.3.2. Branje in izluščevanje podatkov

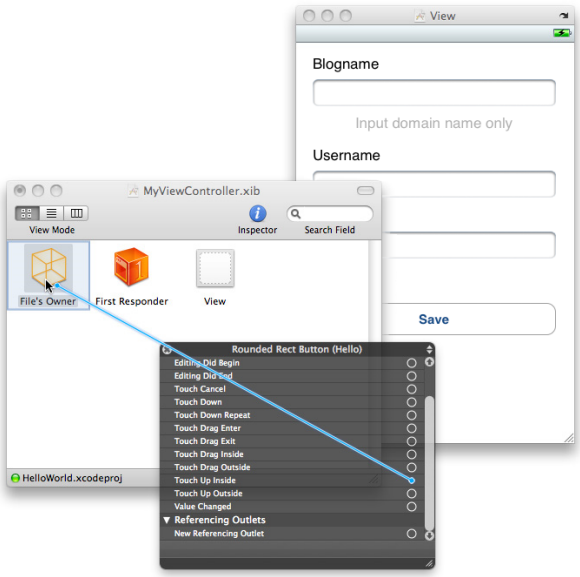
Branje in izluščevanje je v veliki meri podobno načinu, ki sem ga uporabil pri razvoju za Android platformo. Izluščevalnik je dogodkovno usmerjen in proži metode `didStartElement()`, `foundCharacter()` in `didEndElement()`, znotraj katerih obravnavamo posamezne XML elemente in njihovo vsebino. Od načina uporabljenega pri Android-u se razlikuje v tem da uporablja ključ-vrednost način kodiranja *(angl. Key-Value Coding)*, kar pomeni da se lahko v veliki meri izognemo prirejanju spremenljivk, če so te enako poimenovane kot elementi v XML datoteki. Tak način lahko občutno poenostavi implementacijo.

Spodaj je podan enostaven primer, ki demonstrira razliko med običajnim in ključ-vrednost načinom kodiranja

```
// Primer običajnega kodiranja
- (id)tableView:(NSTableView *)tableview
      objectValueForTableColumn:(id)column row:(int)row
{
    ChildObject *child = [childrenArray objectAtIndex:row];
    if ( [[column identifier] isEqualToString:@"name"] ) {
        return [child name];
    }
    if ( [[column identifier] isEqualToString:@"age"] ) {
        return [child age];
    }
    if ( [[column identifier] isEqualToString:@"favoriteColor"] ) {
        ...
    }
    ...
}

// Primer Key-Value kodiranja
- (id)tableView:(NSTableView *)tableview
      objectValueForTableColumn:(id)column
      row:(int)row
{
    ChildObject *child = [childrenArray objectAtIndex:row];
    return [child valueForKey:[column identifier]];
}
```

4.4.3.3. Prikaz prispevkov

Izdelovanju pogledov za iPhone je namenjen zmogljiv program za gradnjo uporabniških vmesnikov, ki je vgrajen v razvojno okolje. Omogoča nam, da uporabniški vmesnik sestavimo iz osnovnih elementov, kot so tabele, seznami, gumbi, oznake, navigacijski elementi, itn. na principu "odvleci in spusti". Medtem ko moramo pri razvoju za Android in WebOS platformo v kodi implementirati poslušalce *(angl. listeners)* dogodkov in primerno reagirati, je pri iPhone-u potrebno sprogramirati funkcijo, ki se bo zagnala ob dotiku in jo v graditelju uporabniškega vmesnika povezati z gumbom. Za vse ostalo poskrbi sistem.



Slika 15: Izdelava uporabniškega vmesnika

Način ustvarjanja je za začetnika zapleten, saj je težko preprosto ugotoviti, kaj je potrebno povezati s katerim elementom, mimogrede pa pozabimo kakšno povezavo.

4.4.3.4. Upravljanje z nastavitvami aplikacije

Izdelava mehanizma za shranjevanje aplikacijskih nastavitev je neodvisna od implementacije aplikacije in delovanja aplikacije. Gre za ločeno datoteko Root.plist, ki jo samodejno ustvari orodje XCode, ko z uporabo čarovnika v projekt vstavimo skupek nastavitev *(angl. Settings Bundle)*. Orodje prav tako tvori o datoteke potrebne za lokalizacijo nastavitev.

Key	Type	Value
Root	Dictionary	(3 items)
Title	String	iBlogger
StringsTable	String	Root
PreferenceSpecifiers	Array	(3 items)
Item 1	Dictionary	(5 items)
Title	String	Blog Name
Type	String	PSTextFieldSpecifier
KeyboardType	String	Alphabet
Key	String	blogName
DefaultValue	String	
Item 2	Dictionary	(5 items)
Title	String	User name
Type	String	PSTextFieldSpecifier
KeyboardType	String	Alphabet
DefaultValue	String	
Key	String	userName
Item 3	Dictionary	(6 items)

Slika 16: Tabelarični prikaz elementov za shranjevanje nastavitev

Elemente nastavitev ustvarimo ročno s pomočjo tabele in si pomagamo osnutkom, ki ga tvori ogrodje. Manjša težava je v tem, da moramo lastnosti elementov vpisovati ročno in jih ni moč

izbrati iz padajočega menija. To zna biti problem predvsem v času, ko se privajamo na okolje, saj še ne moremo vedeti katere vrednosti so možne.

Ko vnesemo vse potrebne elemente v tabelo so naše nastavitve vidne v nastavitveni aplikaciji. To je nekakšen repozitorij za nastavitve vseh aplikacij, ki so nameščene na napravi. Če želimo nastavitve uporabljati znotraj naše aplikacije, pa je potrebno izdelati uporabniški vmesnik za prikaz ter krmilnik za dostop.

V kodi dostopamo do nastavitev na sledeči način:

```
NSString blogName = [[NSUserDefaults standardUserDefaults]
                      stringForKey:@"blogName"];

fieldBlogName.text = blogName;
```

Mehanizem za shranjevanje nastavitev lahko implementiramo izredno hitro, v primeru da ne potrebujemo nastavitev znotraj aplikacije. V nasprotnem primeru pa moramo izdelati pogled in krmilnik, kar je zamudno opravilo.

4.5. Primerjava razvoja in ugotovitve

	Zahtevnost programskega jezika	**Dokumentacija**	**Razvojno okolje**	**Dostop do funkcionalnosti naprave**	**Čas potreben za izdelavo**
Android	Srednja	8	10	9	7
WebOS	Nizka	5	3	6	3
iPhone	Visoka	9	7	8	10

Tabela 6: Primerjalna tabela razvoja

Kar se programskih jezikov tiče, sem imel z Javo in JavaScript izkušnje še pred začetkom diplomskega dela. Predznanje mi je koristilo pri sami implementaciji, vendar se je bilo potrebno spoznati s specifičnimi programskimi vmesniki za platformo. Stopanja poznavanja jezika, je v veliki meri odvisna od izkušenj, vendar je dejstvo, da je JavaScript najlažji programski jezik med naštetimi. Objektni C po drugi strani zahteva veliko programske discipline. Kljub temu pa bo nekomu, ki ima izkušnje s pisanjem aplikacij v tem jeziku, pomenil razvoj za iPhone enak izziv. Ne glede na zahtevnost programskega jezika, sem se v vseh primerih precej oziral na dokumentacijo. Apple nudi celovito in obsežno dokumentacijo, ki bo začetnika seznanila z jezikom, načini razvoja in programskimi vmesniki. Pri Android-u je bolj usmerjena na lastnosti platforme, medtem ko je pri Palm-u še skromna in neoblikovana, a kljub temu spodobna. Pri razvojnem okolju sem pričakoval največ razlik, a so bile v končni fazi omejene na sposobnost detekcije in popravljanja napak v kodi. Vsa orodja so zmogljiva in prilagojena razvoju za platforme, vendar ima Eclipse in dodatek za razvoj Android aplikacij veliko prednost pred ostalimi in s tem dober potencial za privabljanje novih

razvojalcev. Čas potreben za razvoj aplikacij za posamezno platformo je zelo težko oceniti. Dejstvo je, da sem zaradi poznavanja Java, JavaScript in okolja Eclipse porabil manj časa, kot pri razvoju v objektnem C in XCode okolju. Vseeno pa je opaziti, da je bil razvoj v JavaScript za Palm Pre nahitrejši, predvsem zaradi uporabljene JSON notacije in močnega gradnika za gradnjo seznamov. Razvoj za iPhone je bil najdaljši zaradi učenja objektnega C in večje pozornosti, ki jo je potrebno posvetiti upravljanju s pomnilnikov. Pri Android platformi in WebOS tega zaradi samodejnega zaseganja in sproščanja virov to ni potrebno.

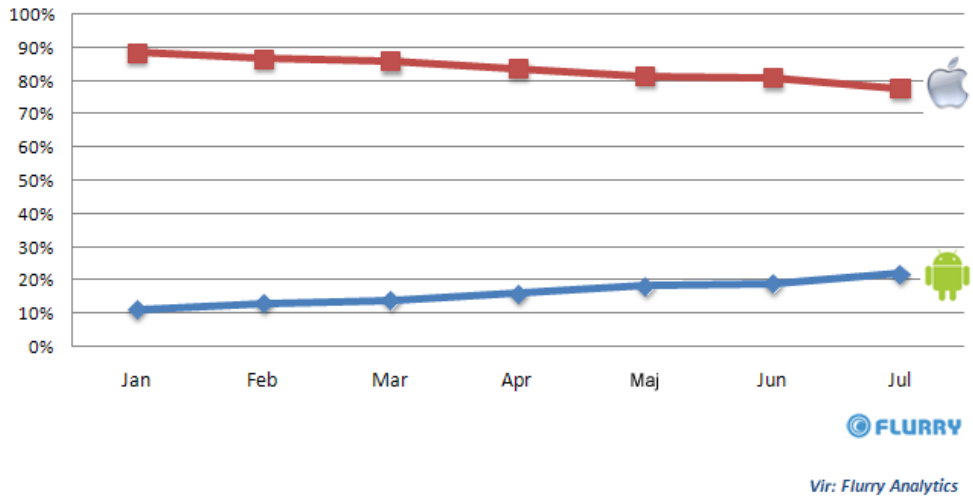
Google je platformo Android zelo dobro zasnoval. Ponudil je izredno močno okolje za razvoj aplikacij. Java je v tem trenutku najbolj razširjen programski jezik, s čimer so si zagotovili pripadnost precej velike razvijalske skupnosti. Eclipse je zelo zmogljivo orodje za razvoj v Javi, poleg tega pa so izdali dodatek, ki se popolnoma integrira v okolje in precej poenostavi razvoj, testiranje in razhroščevanje aplikacij za platformo. Zaradi odlične korekcije napak, čarovnikov in nastavkov je primerno okolje za razvijalca, ki nima izkušenj z razvojem. Google je prav tako ponudil ogromno fleksibilnih programskih vmesnikov za dostop do funkcionalnosti naprave in z uporabo mehanizma namer programerjem omogočil izdelavo aplikacij, ki so tesno povezane z delovanjem naprave.

Palm WebOS se je po drugi strani z JavaScript, HTML in CSS še bolj približal razvijalcem. Danes je namreč ogromno razvijalcev spletne vsebine, ki uporabljajo takšno tehnologijo. Razvoj enostavnejših aplikacij je izredno hiter, saj ogromno dela opravijo dobro zasnovani programski vmesniki, tako da razvijalcu ni potrebno skrbeti za podrobnosti, kot je primer pri razvoju za Android in iPhone platformo. Slabost je premalo zmogljivo razvojno okolje, oziroma dodatek zanj in zaenkrat omejen dostop do funkcij naprave preko, sicer zmogljivih, programskih vmesnikov.

iPhone mi je med razvojem povzročal največ težav, predvsem zaradi programskega jezika s katerim sem se prvič srečal ravno v okviru diplomskega dela. Potrebno je bilo nekaj časa preden sem spoznal način pisanja v objektnem C in se lahko posvetil razvojnemu okolju ter razvoju aplikacije. Okolje je izredno zmogljivo, vendar ne omogoča takšne integracije z napravo, kot Android platforma. Kot prednost bi izpostavil dobro orodje za izdelavo uporabniških vmesnikov in analizo zmogljivosti aplikacije. Slabost je omejitev razvoja na MacOS X operacijskih sistemih in strma krivulja priučitve programskega jezika.

Na primeru aplikacij razvitih za tri različne platforme sem ugotovil, da kljub vsemu, programski jezik, razvojno orodje in različni arhitekturni stili operacijskih sistemov, nimajo bistvenega vpliva na razvoj aplikacij. Z dobro mero programerske discipline in volje se je moč priučiti razvoja za vsakršno platformo. Za vsak operacijski sistem lahko izdelamo zmogljivo aplikacijo, od razvijalčevega poznavanja sistema, pa je odvisen čas in kvaliteta končnega izdelka. Predznanje programiranja v nekem jeziku je torej lahko le prednost za izbiro platforme. Večji problem je izbrati pravo platformo za razvoj v prihodnosti, saj se bo

šest operacijskih sistemov le s težka obdržalo v tehnološkem ekosistemu. Na sliki lahko opazimo pojenjajoči trend razvoja novih aplikacij za iPhone in rastoči trend razvoja za Android.



Slika 17: Trend razvoja novih aplikacij za iPhone in Android platformi

5. Sklepne ugotovitve

Razvoj za pametne telefone je v polnem zamahu. Danes skoraj vsi proizvajalci mobilnih naprav na trgu ponujajo vsaj en pameten telefon. Google je s platformo Android to omogočil tudi tistim, ki se v razvoj lastnega operacijskega sistema niso spustili. Tako proizvajalci pametnih telefonov, kot razvijalci so ga zelo dobro sprejeli. iPhone zaradi ogromne baze uporabnikov, kljub zahtevnemu programskem jeziku še vedno privablja nove razvijalce, Palm Pre, pa se skuša v igro pridružiti predvsem z enostavnim modelom razvoja.

Razvoj za pametne telefone bo v prihodnosti še bolj priljubljen, saj naprave postajajo dostopnejše, orodja za razvoj pa popolnejša. Čeprav sta za razvoj potrebna disciplina in izkušnje, sta za enostavnejše aplikacije dovolj poznavanje konceptov programiranja ter volja. Za razvoj namreč obstajajo zmogljiva orodja, ki stopnjo potrebnega znanja v programiranju precej znižajo, s tem da predlagajo možne rešitve, pripomorejo pri gradnji kode in preko čarovnikov ustvarijo nastavke za metode in funkcije. Problem ni več kako razvijati, temveč na kateri sistem se osredotočiti za razvoj v prihodnosti. Ugotovil sem, da imajo vse tri platforme velik potencial za prihodnost, vseeno pa menim, da bo zaradi celovitosti, prilagodljivosti in možnosti delovanja na raznih napravah platforma Android v prihodnosti najbolj uspešna.

Na podlagi zgoraj omenjene ugotovitve sem za nadaljnje raziskovanje in delo izbral razvoj za Google-ovo platformo Android. V okviru diplomskega dela sem namreč z razvojem aplikacije za urejanje spletnih dnevnikov izkoristil le del sposobnosti platforme Android. Ob tem pa sem spoznal, da je platforma zrela za razvoj in distribucijo zahtevnejših aplikacij, ki jih nameravam razviti v prihodnosti.

Slike

Sheme

Tabele

Viri

[1] (2006) Augumented Reality Page. Dostopno na: http://www.se.rit.edu/~jrv/research/ar/

[2] (2006) The dawning age of ubiquitous computing. Dostopno na: http://www.studies-observations.com/everyware/samples/everyware_intro.pdf

[3] (1996) Ubiquitous Computing. Dostopno na: http://sandbox.xerox.com/ubicomp/

[4] (2009) A closer look at the LG Watch Phone. Dostopno na: http://ces.cnet.com/8301-19167_1-10137452-100.html

[5] (2009) IBM – Cloud Computing. Dostopno na: http://www.ibm.com/ibm/cloud/

[6] Virtualization and cloud computing (in Slovene). Dostopno na: http://www.slideshare.net/tomirozman/dv2009trozmancloudvirt

[7] (2009) Smartphone – Wikipedia. Dostopno na: http://en.wikipedia.org/wiki/Smart_phone

[8] (2009) Android Smartphone Shipments to Grow 900 Percent in 2009. Dostopno na: http://www.strategyanalytics.com/default.aspx?mod=pressreleaseviewer&a0=4728

[9] (2009) Android Market Sales, Are Those Tears or is it Raining in Here? Dostopno na: http://larvalabs.com/blog/iphone/android-market-sales/

[10] (2009) TIOBE Programming Community Index for September 2009. Dostopno na: http://www.tiobe.com/index.php/content/paperinfo/tpci/index.html

[11] (2008) iPhone Developers Go From Rags to Riches. Dostopno na: http://www.wired.com/gadgetlab/2008/09/indie-developer/

Izjava o samostojnosti dela

Izjavljam da sem diplomsko delo izdelal samostojno pod vodstvom mentorice doc. dr. Mojce Ciglarič.

Ljubljana, 10.9.2009 Saša Nebojša Potežica